Сергей Дунаев

Технологии Интернет-программирования

Дюссельдорф • Киев • Москва • Санкт-Петербург

УДК 681.3.06

Книга представляет собой практическое руководство и подробный путеводитель по самым современным и бурно развивающимся Интернет-технологиям и программным средствам. Современная программа — это не монолитный кусок кода, как раньше, а скорее всего — управляемый сценарий, приводящий в действие множество информационных объектов, существующих в виде апплетов, скриптлетов, сервлетов, XML-конструкций, JSP- и ASP-страниц и т. д. Приведенные примеры демонстрируют, как надо правильно применять две ведущие технологии Интернета: JavaBeans/CORBA и ActiveX/DCOM. Используются два основных языка программирования Visual Basic и Java, хотя имеются примеры и на C++, Pascal, LotusScript. Активно обсуждается связь Java и XML.

Для специалистов в области электронной коммерции и разработки Интернет-приложений

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Дунаев С. Б.

Технологии Интернет-программирования. — СПб.: БХВ-Петербург, 2001. — 480 с.: ил.

ISBN 5-94157-086-4

© С. Б. Дунаев, 2001

© Оформление, издательство "БХВ-Петербург", 2001

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.05.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 4000 экз. Заказ 1009
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9-я линия, 12.

Содержание

Введение **11**

Глава 1. Что минимально нужно знать разработчику об ActiveX и DCOM **17**

Что такое COM и DCOM 18
Пример создания объектно-распределенного приложения с использованием DCOM 23
Что подразумевается под технологиями ActiveX 27
Взгляд на некоторые особенности технологии 27
Управляющие элементы ActiveX (ActiveX Controls) 27
Создание и регистрация управляющих элементов ActiveX 29
Контейнер и взаимодействие с ним 34
Свойства окружения 34
Дополнительные свойства 35
Объект *UserControl* 35
Ключевые события 35
Создание программного интерфейса элемента управления ActiveX 36
Свойства 36
Сохранение и восстановление состояния объекта 37
Методы 37
События 37
Размещение управляющих элементов ActiveX на Web-страницах 40
Элемент управления Popup Menu 43
Инсталляция и распространение управляющих элементов ActiveX 46
Лицензирование 46
ActiveX-библиотека (ActiveX DLL) 49
Ключевые понятия: in-process и out-of-process 50
ActiveX-сервер (ActiveX EXE) 51
Пример взаимодействия Oracle и Notes с использованием технологий ActiveX 52
Реализация интерфейса подключения внешних процедур OLE Automation External Procedure APIs 53
Объектно-ориентированное программирование в Oracle 54
ActiveX-документы и документы-контейнеры типа MS Internet Explorer 56
Обогащение OLE-технологий на новом витке спирали развития Интернета 57
Активная платформа и ActiveX 62
Клиентская технология ActiveX (Active Desktop) 63
Серверная технология ActiveX (Active Server) 64

Дополнительные средства от Microsoft для создания распределенных приложений ... 65
Возможности Microsoft Transaction Server ... 65
Возможности Message Queue Server ... 70
Приложение к главе 1. Интернет-литература, бесплатные ресурсы и источники ... 77

Глава 2. Что минимально нужно знать разработчику о JavaBean ... 85

В чем суть архитектуры JavaBean ... 85
Что такое Bean (Боб)? ... 89
Инспекция свойств и событий ... 94
Использование компонентов Bean в качестве ActiveX-элементов ... 96
Мост JavaBean-ActiveX ... 96
Преобразование JavaBean в ActiveX ... 96
Кратко о главном ... 96
Пакетирование и регистрация ... 97
Шаг 0. Старт упаковщика ... 97
Шаг 1. Указание местонахождения jar-файла ... 97
Шаг 2. Выбор компонента JavaBean ... 97
Шаг 3. Присвоение имени ActiveX ... 97
Шаг 4. Указание принимающего каталога ... 99
Шаг 5. Начало генерации ... 99
Опции командной строки ... 100
Подготовка к встраиванию Bean в программу на Visual Basic ... 100
Импортирование JavaBean в Visual Basic ... 102
Прямое соединение в Internet Explorer ... 106
OLE Automation Server ... 106

Глава 3. CORBA и Enterprise JavaBeans — лидеры новейших технологий ... 109

Зачем нужна CORBA ... 109
Распределенные приложения ... 113
Распределенные данные ... 114
Распределенные вычисления ... 114
Распределенные пользователи ... 114
От клиента к серверу ... 115
Клиентская часть CORBA ... 116
Серверная часть CORBA ... 117
Сервисы CORBA ... 117
CORBA-продукты ... 118
Объектные Web-модели ... 119
Взаимодействие типа клиент-сервер в объектной модели Web ... 121
Имеются ли другие игроки на этом рынке ... 122
Примеры с использованием CORBA ... 124
Приложение *Акции* ... 124
Программа Hello World! ... 134
В чем преимущества Enterprise JavaBeans (EJB) ... 145
Параллельные миры ... 145
Краткие сведения о EJB ... 147
Для чего создана архитектура EJB ... 147

Примеры использования EJB ... 148
Поддержка разнообразных клиентов ... 148
Повторно используемые JavaBeans ... 148
Механизм наследования и кастомизация ... 148
Архитектура EJB ... 149
Сервер Enterprise JavaBeans ... 149
EJB-контейнер ... 149
Инсталляция Enterprise Beans в EJB-контейнере ... 151
Как работает EJB ... 151
Изменяемые и неизменные объекты ... 153
Jar-файл EJB ... 155
Чем Enterprise JavaBeans отличаются от сервлетов ... 155
Краткие сведения о J2EE ... 155
Компонентные технологии ... 156
Технологии обслуживания (Service Technologies) ... 157
Технологии управления сообщениями (Messaging Technologies) ... 158
J2EE — открытая платформа разработок на Java ... 159
Как J2EE стыкуется с EJB ... 162
J2EE-платформа и EJB-архитектура ... 162
Примеры использования ... 162
Исходный код HTML-страницы ... 163
Исходный код сервлета ... 164
Создание сеансового Bean ... 166
Компоненты приложения ... 166
Компиляция сеансового Bean ... 168
Компиляция сервлета ... 168
Старт сервера приложений J2EE ... 169
Запуск среды развертывания приложения ... 169
Компоновка распределенного приложения ... 170
Выполнение распределенного приложения J2EE ... 171
Приложение к главе 3. Интернет-ресурсы и источники ... 172
Книги и статьи ... 172

Глава 4. Микропрограммы для браузеров. Удаленное и локальное исполнение программ в Интернете/интранете ... 173

Апплеты ... 173
Виртуальная машина Java ... 174
Изменение исполняющей Java-машины по умолчанию ... 177
Пример апплета для работы с буквами ... 179
Сервлеты ... 189
Java-сервлеты ... 189
Сервлеты на базе Apache ... 200
Аглеты ... 201
Что такое аглет и Java Aglet API (J-AAPI) ... 201
Главные цели создания ... 202
Архитектура аглетов ... 202
Скриптлеты ... 211
Сначала немного о скриптах ... 211
Что может VBScript ... 211

Как VBScript встраивается в HTML-документ ... 211
Экспресс-описание VBScript ... 212
DHTML-скриптлеты ... 215
XML-скриптлеты ... 217
Архитектура XML-скриптлетов ... 218
Примеры XML-скриптлетов ... 220
Java-скриптлеты ... 222
ASP-сценарии ... 223
Распределенная архитектура Windows DNA ... 224
Архитектура Web-сервера Microsoft IIS ... 224
Объектный интерфейс Microsoft на базе OLE DB — ADO ... 227
ODBC, OLE DB и ADO ... 228
Объектная модель ADO ... 229
ASP-примеры ... 230
Простая объектная модель ... 234
Опции объекта *Recordset* ... 234
Модифицирование данных ... 242
JSP-сценарии ... 253
Что такое JavaServer Pages ... 253
Модели доступа к JSP ... 254
Пример использования JSP-файла (с упрощенным синтаксисом IBM WebSphere) ... 255
Пример сервлета (PopulateBeanServlet) ... 256
Пример HTML-формы ... 257
Пример результирующей JSP-страницы (DisplayData.jsp) ... 257
Что могут содержать JSP-файлы ... 258
HTML-теги ... 258
Общепринятый JSP-синтаксис (JSP-спецификация) ... 259
JSP-API и примеры стандартных JSP-сценариев ... 262
Средства для создания JSP-файлов ... 264
IBM WebSphere Studio Version 3.0 ... 264
NetObjects ScriptBuilder ... 265
IBM VisualAge for Java Enterprise 3.0 ... 265
Главная цель изобретения JSP ... 265
Использование JavaBean-компонентов в JSP ... 266
Для серверов приложений на основе JSWDK ... 266
Использование апплетов в JSP (plugin) ... 271
Встроенный Java-код (scriptlet) ... 276
Совместное использование JSP и XML ... 277
DOM (Document Object Model) ... 279
Интернационализация и проблемы с кириллицей по отношению к JSP ... 282
Листинг 1. Message.properties ... 283
Листинг 2. Message_ru_RU.properties ... 284
Методы загрузки Java-кодов, сервлетов и апплетов ... 285
Нововведения в концепции апплетов и загружаемого по сети ПО со стороны Microsoft ... 285
Инновации Microsoft ... 285
Файлы OSD (Open Software Description) ... 287
Примеры использования ... 288

Безопасность в Интернете — подписанные апплеты ... 290
Методы фирмы Sun, предлагаемые для распространения по сети загружаемого и безопасного программного обеспечения ... 290
Приложение к главе 4. Интернет-ресурсы и источники ... 294
Протоколы Интернета ... 294
Безопасность ... 294
Скриптлеты ... 294
Сервлеты ... 295
Книги и статьи ... 296

Глава 5. Web-серверы в качестве серверов приложений ... 299

Открытые системы и бизнес-решения ... 299
Архитектура ... 299
Клиенты ... 299
Сетевая инфраструктура ... 300
Компонентная интеграция ... 300
Программные средства для создания Web-приложений ... 301
Бизнес-приложения ... 301
Лучшие Web-серверы и серверы приложений ... 303
Серверы-приложений на базе Java и Apache ... 303
Jakarta ... 303
Tomcat ... 304
Последовательность действий по установке Tomcat из VisualAge for Java ... 305
Сервер приложений IBM WebSphere 3.0 ... 321
WebSphere Studio 3.0 ... 323
WebSphere Application Server Standard Edition ... 324
WebSphere Application Server Advanced Edition ... 325
WebSphere Application Server Enterprise Edition ... 326
Элементы WebSphere-приложений ... 327
Java-сервлеты ... 327
Java Server Pages ... 328
Enterprise JavaBeans ... 330
Основные шаги при использовании EJB ... 330
Lotus Domino и WebSphere ... 331
Требования для инсталляции ... 332
Основные шаги ... 332
Регистрация пользователя ... 332
Конфигурирование IIS для Domino ... 335
Поддержка сервлетов IIS для WebSphere ... 338
Конфигурирование IIS для WebSphere ... 338
Сервер приложений Oracle iAS ... 339
Что такое Oracle iAS ... 339
Коммуникационные сервисы ... 340
Oracle HTTP Server ... 340
Модули Oracle HTTP Server ... 341
Сервисы представлений ... 342
Oracle Portal ... 342
Apache JServ ... 343
Oracle JSP ... 343

PL/SQL Server Pages ... 344
Perl-интерпретатор ... 344
Бизнес-логика ... 345
Бизнес-компоненты для Java в Oracle ... 345
Поддержка расширенной модели Java ... 345
Oracle XML Developer's Kit ... 346
Программно-аппаратная масштабируемость Oracle ... 346
Протокол SOAP ... 347
Приложение к главе 5. Ресурсы и ссылки в Интернете ... 348
Ресурсы ... 348
Ссылки ... 349

Глава 6. Способы связи с базами данных из Интернета и Web-приложений, основанные на традиционных методиках ... 351

Концепция ODBC ... 351
Зачем нужен ODBC ... 351
ODBC Driver Manager ... 352
Построение стандартных ODBC-приложений в Microsoft Windows 95/98/NT ... 353
Различные стандарты ODBC ... 358
ODBC-решения ... 358
Архитектура ODBC ... 359
Функциональные возможности стандарта ODBC 3.x ... 360
Доступ к базам данных из Web-ориентированных Java-апплетов и Java-программ ... 363
Java-программы и апплеты с интерфейсом JDBC-ODBC ... 363
Специально ориентированные Java-приложения ... 370
RMI-приложения ... 370
Инкапсулированный в СУБД Java ... 377
Новые возможности работы с таблицами с использованием JFC-классов ... 378
Модуль JDBCAdapter.java ... 385
Модуль TableMap.java ... 389
Модуль сортировки TableSorter.java ... 391
Приложение к главе 6. Интернет-ресурсы и источники ... 400

Глава 7. Java и XML как новый способ работы с данными и СУБД ... 403

Зачем нужен XML ... 403
Суть XML ... 405
Многоуровневые приложения на основе Java и XML ... 407
Клиентские приложения ... 408
Серверные приложения ... 409
Какие бывают парсеры ... 414
Суть DOM ... 415
Суть SAX ... 416
Когда использовать DOM ... 416
Когда использовать SAX ... 417
Пример DOM ... 417
Компиляция программы ... 419
Выполнение программы ... 419
Результат выполнения программы ... 419

Пример SAX ... 420
Текст программы (http://java.sun.com/xml) ... 420
Исходный файл book-order.xml ... 422
Компиляция программы ... 423
Выполнение программы ... 423
О DOM и SAX API ... 424
Использование XML в Oracle ... 424
Oracle8*i* и *inter*Media ... 424
Сохранение XML-документов как обычных документов ... 425
Сохранение XML-документов в качестве данных ... 425
XML-парсеры в Oracle ... 425
Поддержка XSL-преобразований ... 425
Поддержка Namespace ... 426
Пример Java XML-парсера ... 426
XML-генератор классов для Java ... 428
Пример использования XML-генератора классов ... 429
Утилита XML SQL для Java ... 433
Генерация XML из результатов запроса ... 434
Пример генерации XML из результирующего набора ... 435
Пример генерации XML из результатов запроса и структурирования данных ... 435
Запись XML в таблицу базы данных ... 436
XSQL Servlet ... 437
Пример использования XSQL Servlet ... 439
Использование XML в Lotus Domino Server ... 441
Сервер приложений Domino — лучшая XML-платформа ... 443
LotusXSL-процессор ... 444
Как используют XML в Domino ... 446
Использование видов для создания XML ... 446
Создание XML-кода из простого вида при помощи формул Lotus ... 448
Использование Domino-форм для создания XML ... 449
Создание XML при помощи агентов LotusScript или Java ... 451
Экспорт данных из базы Domino в XML-формате ... 452
Сетевой импорт XML-данных в базу данных Domino ... 454
Использование Java-сервлетов для генерации XML ... 454
Как Lotus использует XML в своих собственных продуктах ... 456
Апплеты Domino для обеспечения пользовательского интерфейса ... 456
Файловые форматы в SmartSuite и eSuite ... 457
Резюме ... 457
Microsoft и XML ... 457
BizTalk Editor ... 460
BizTalk Mapper ... 462
Приложение к главе 7. Интернет-ресурсы и источники ... 463

Глава 8. Что такое GNU, Open Source и Интернет-сообщество программистов ... 465

Введение

За последние 30 лет произошли настолько грандиозные изменения в мире компьютерной техники и программного обеспечения, что даже прославленные ветераны иногда не могут понять, в чем смысл преобразований, связанный с Интернет-стандартами, и почему в наше время навязывание традиционных методов проектирования и программирования может привести к неизбежным проблемам и серьезным последствиям. Например, Роджер Прессман — автор многих книг по программированию и редактор колонки в IEE SoftWare признается: "За 30 лет своей деятельности в сфере проектирования программного обеспечения я четко уяснил одну вещь: разработчики — всегда заложники модных течений. Появляется новая многообещающая технология, и мы бросаем все (по крайней мере пытаемся) и отдаемся ей полностью, начисто забывая уроки прошлого. Последняя такая "горячая" новинка — разработки на базе Интернета" (см. журнал "Открытые Системы" № 5 от 6/99). Этим характерным высказыванием Р. Прессман ясно дает понять, что круто изменившаяся парадигма программирования вызывает у него неприятие, раздражение и отталкивающее непонимание. Растерянность, воцарившаяся в головах многих разработчиков и идеологов традиционных систем, вполне понятна, если учесть взрывоподобные и всесокрушающие темпы компьютеризации общества и глобализацию мировых компьютерных ресурсов и идей. Сегодня компьютер — это сеть. В этом нет уже ничего революционного, это данность. В ближайшем будущем мы будем сталкиваться только с распределенными приложениями, куски которых будут работать на совершенно различных компьютерах, гетерогенных сетях, в различных операционных системах с использованием различных языков, технологий и запоминающих устройств. Компьютер уже сейчас начинает терять свой традиционный облик и подобно легендарному Протею способен принимать самые различные формы, начиная от вмонтированных в стены кубических блоков до миниатюрных наручных часов и карманных видеотелефонов. В программном обеспечении отделение интерфейсов (что?) от алгоритмов (как?) будет достигнуто через кроссплатформность, кроссязыки, кросс-проектирование.

Монолитные приложения — это приложения, где доступ к данным, бизнес-логика и представление данных на экране компьютера "вшиты" в одну программу. Такие приложения сейчас уже почти никто не использует. Для исполнения этих примитивных одноуровневых, но весьма сложных и ресурсоемких приложений в свое время существовали мэйнфреймы, где одновременно функционировало множество копий подобных задач, выполняющихся в многозадачной среде со сложными процессами организации вычислительного процесса и специальным языком управления заданиями.

Клиент-серверные приложения — в настоящее время большинство работающих приложений построено на базе именно этой технологии. Естественно, что за длительное время своего существования данная технология претерпела значительные усовершенствования. От файловых серверов NetWare произошел постепенный откат с заменой их на кроссплатформные масштабируемые серверы приложений с установленным на них многофункциональным программным обеспечением Oracle, IBM или Sybase. Файловые серверы, впрочем, остались для обеспечения совместимости с наследуемыми системами и сохранения архивов и промежуточных результатов (рис. 1).

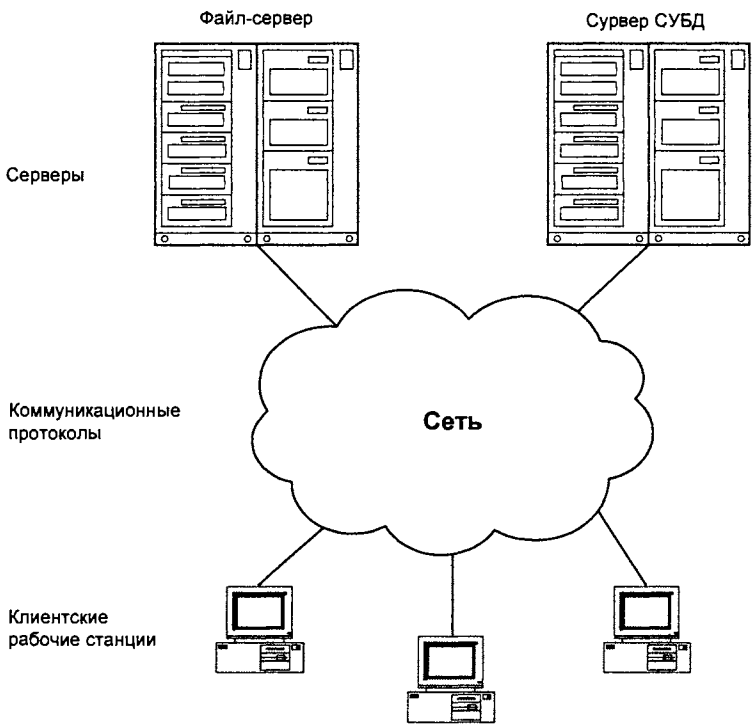


Рис. 1. Клиент-серверная архитектура

Распределенные приложения. Используя объектно-ориентированные средства программирования и объектно-ориентированные средства разработки, мы можем получить гораздо более надежные, адаптируемые к любым условиям, дешевые, способные к быстрым изменениям кроссплатформные программы, которые очень легко создавать и эксплуатировать. Задачи рынка программного обеспечения сегодня таковы, что разработчики никогда не знают точно, как будет определена и специфицирована задача в деталях, потому что их заказчики сами не могут ясно сформулировать свои требования; что, по большому счету, не имеет особого значения, поскольку как только система будет завершена, она уже может устареть и потребовать новых модификаций, которые становятся перманентными в условиях динамично обновляющейся конъюнктуры рынка. В идеале, программы должны стать как бы живыми и самообучающимися, они должны уметь приспосабливаться к возмущениям извне и работать не с данными, а с метаданными, т. е. их описаниями, в которые могут вноситься постоянные изменения. В средствах нового проектирования модель водопада больше не работает, спиральная модель чаще всего оказывается непригодной и ставка делается на динамическое определение типов ("прототипирование"). Объектно-ориентированный подход позволяет интегрировать старые и новые технологии, используя "обертку" (wrapping) и инкапсуляцию. К примеру, можно рассматривать модули, написанные в процедурных языках, как объекты и использовать их. Инкапсуляция хороша тем, что ее можно применить для интеграции и реинжиниринга наследуемых систем, сохраняя старую и опробованную функциональность. Для этого нужно упрятать старые модули внутрь новой объектной модели и обращаться к ним через объектные интерфейсы. Хорошим примером в этой области является разработка интеллектуальной поисковой машины IBM Intelligent Miner. Там реализован принципиально новый блок интерфейсов JavaGUI, который использует набор интерфейсов C-API для обращения к ядру, написанному ранее на C++. Пользователь может работать с новым набором графических Java-интерфейсов, которые, в свою очередь, обращаются к старому ядру через C Wrapper. Пользователь, однако, даже не чувствует этого, потому что имеет дело только с Java. Другим важным преимуществом объектно-ориентированного подхода является повторно используемый код. Распределенные объекты могут работать либо независимо, либо в контейнере. Каждый модуль или компонент может взаимодействовать с другими компонентами. Разработчики не создают новое приложение с нуля. Вместо этого они создают гораздо более мелкие и вполне понятные объекты, а приложение состоит из этих объектов и активно использует уже существующий код. Поскольку большая часть кода распределенного приложения работает на серверах, отпадает необходимость, в случае модификации кода, массированного обновления клиентской части приложения, которое подкачивает измененный код в автоматическом режиме и тем самым модифицирует само себя. На рис. 2 показаны 3 уровня многоуровневой распределенной архитектуры.

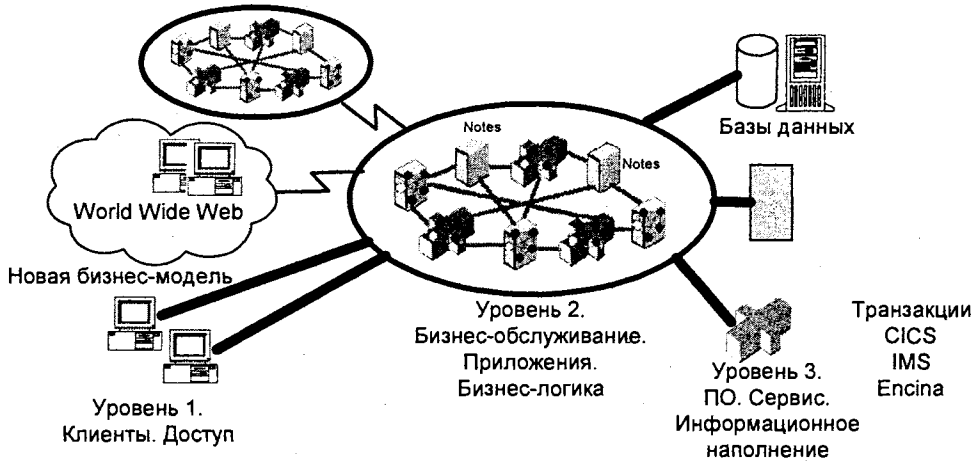


Рис. 2. Многоуровневая распределенная архитектура

Объектно-ориентированный подход чрезвычайно хорош при создании динамичных Web-приложений, к которым ежедневно обращаются несколько тысяч клиентов с самыми различными запросами. Это может быть Web-магазин, Web-банк, Информационный ежедневник или просто поисковый сервер с громадной, постоянно обновляемой базой данных. Важно спроектировать и построить как его инфраструктуру, так и процесс модификации его информационного содержимого. Построение инфраструктуры — это все то же проектирование программного обеспечения только новыми инструментальными средствами. С другой стороны, построение процесса модификации содержимого похоже на технологию подготовки и выпуска крупного информационного ежедневника. В этот процесс необходимо включить такие действия, как ввод данных, проверку правильности данных и т. д. Для решения такой задачи требуется либо большое число операторов, либо значительная автоматизация. Обе эти задачи могут сильно отличаться количеством и свойством используемых методов. При этом следует учитывать, что при Интернет-проектировании часто отказываются работать такие традиционные вещи, как поэтапное проектирование, планирование сроков выпуска версий, тестирование и т. д. Может быть поэтому некоторые разработчики и консультанты все чаще говорят о "новой бизнес-модели" для программного обеспечения, в которой Web-система становится арендуемой системой с почасовой оплатой, а не продуктом, который пользователь покупает и которым владеет.

Хостинг и хакинг. Проблема специалистов. В новое время безудержный энтузиазм пионеров компьютерной индустрии, благодаря которому мы сегодня

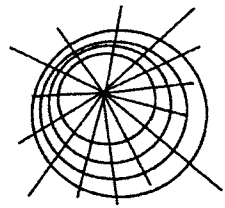
имеем все, о чем и не смели мечтать 10—15 лет назад, к сожалению, безвозвратно уходит в прошлое. Бесконечный рост невозможен. Бизнес и коммерция посредством новых информационных технологий диктуют свои условия. С течением времени объем работ становится непосильным даже для самых лучших программистов и проектировщиков, поэтому компании, чтобы справиться с требованиями развивающегося рынка, набирают менее компетентных людей, которые, возможно, уже не обладают энергией своих предшественников. Системы на базе Web должны быть спланированы и внедрены в кратчайшие сроки. У молодых разработчиков появился новый термин *хакинг*, т. е сверхбыстрая разработка недокументированных программ, составленных без предварительного плана. Естественно, что при таких темпах требуются совершенно другие инструментальные средства, которые отвечают требованиям сегодняшнего дня. И они появляются. Например, с помощью Lotus Donino Designer можно спроектировать и базу данных, и Web-сайт для Domino-сервера буквально за несколько дней и это будет вполне профессиональный сайт, способный обслуживать до нескольких сотен одновременно работающих с ним пользователей. То же самое можно сказать и об Oracle Tools для Oracle Web Application Server, и о великолепной гамме продуктов Microsoft, и об IBM Web Studio для WebSphere и о других продуктах. Иногда крупные Интернет-провайдеры держат у себя специальную команду проектировщиков и прикладных программистов, которые осуществляют так называемый *хостинг*, т. е. разработку и последующую бессрочную эксплуатацию разработанной системы для крупного заказчика (владельца магазина, брачной конторы, туристического агентства и т. д.) непосредственно у себя на сервере. Следует понимать, что длительность жизни Web-системы может быть весьма недолгой и в то же время, при благоприятном стечении обстоятельств, система может масштабироваться до бесконечности.

В любом случае приходится приспосабливаться к темпам, которые диктует Интернет. Без понимания основополагающих принципов его функционирования и овладения ключевыми технологиями, которые применяются на каждом узле, сейчас просто не обойтись. Чтобы свободно владеть всеми средствами Интернета, нужно знать очень много. Настоящая книга призвана помочь разработчикам находить наиболее эффективные решения, пользуясь широким и разнообразным арсеналом легко доступных инструментальных средств, а непрофессионалам понять — какие возможности открывают для них новые способы проектирования и программирования, чтобы научиться грамотно формулировать перед разработчиками свои задачи. В обоих случаях ясное и краткое описание всех ключевых технологий, проиллюстрированных многочисленными примерами, взятыми из практики построения самых современных систем, должно оказаться полезным. Некоторые главы (например, описание симбиоза Java и XML) посвящены технологиям, уже целиком принадлежащим XXI веку, другие подробно раскрывают новые термины и понятия, такие как *сервлеты*, *аглеты*, *скриптлеты*. В целом, автор

хотел как можно полнее и выразительнее показать ведущие тенденции современного программирования, которое за последнее время, пережив несколько революционных ломок, входит наконец в фазу своего окончательного становления.

Интернет и программирование для Интернета — это не мода, это способ существования и выживания для многих развивающихся и перспективных компаний, благодаря деятельности которых мы уже сегодня можем увидеть контуры совсем недалекого Будущего.

ГЛАВА 1



Что минимально нужно знать разработчику об ActiveX и DCOM

Любому разработчику программного обеспечения в своей повседневной практике постоянно приходится сталкиваться с двумя основополагающими технологиями, без которых сегодня не обходится ни одна инструментальная среда разработки, будь то Delphi, Oracle Developer или MS Visual Studio. Одна из этих технологий носит обобщенное название *ActiveX/DCOM* и распространена на Intel-платформах с семейством операционных систем Windows, а другая параллельная технология называется *JavaBeans/Corba* и не зависит ни от платформы, ни от операционных систем.

ActiveX/DCOM — это корпоративная технология с фирменным знаком Microsoft. Главное ее предназначение — облегчение написания сетевых распределенных объектно-ориентированных приложений. В сущности, *ActiveX* — это ни что иное, как обычный набор библиотек, значительно облегчающих процесс кодирования. Иногда спрашивают, в чем разница между OLE и его составными механизмами (OLE Automation, OLE Documents, OLE Controls) и элементами ActiveX? Ответ очень прост. OLE-механизмы основаны на компонентной объектной модели (COM, Component Object Model), которая позволяет создавать объектно-ориентированные приложения на одной машине в рамках функционирующей на ней операционной системы Windows, а ActiveX использует DCOM (Distributed Component Object Model) — распределенную компонентную объектную модель, которую реализуют библиотеки ActiveX, являющиеся по объему гораздо меньше, чем библиотеки OLE, а по скорости — быстрее. Другими словами, библиотеки OLE переписаны так, чтобы обеспечивать функциональность, достаточную для написания сетевых приложений. Сохранилась и совместимость — любой программный компонент OLE будет работать с библиотеками ActiveX. Модели COM и DCOM опираются на базовые сетевые протоколы, такие как TCP/IP, и входят в состав операционной системы Windows 98/2000, а для более старых реализаций всегда существует возможность инсталлировать эти компоненты из самоустанавливающегося архива (DCOM95.EXE).

Что такое COM и DCOM

COM (Component Object Model) расшифровывается просто — компонентная объектная модель. DCOM (Distributed Component Object Model) — это, соответственно, распределенная компонентная объектная модель.

Microsoft разрабатывает и поддерживает DCOM исключительно для создания серверов приложений и управления распределенными программными объектами. В противовес Microsoft, два других компьютерных гиганта — Sun и IBM — опираются на международный стандарт CORBA. При желании можно совместно использовать обе эти технологии.

DCOM просто расширяет возможности COM в части регистрации удаленных объектов, поэтому его иногда называют "COM с удлиненным проводом".

Прикладные интерфейсы COM API — это все те же, всем известные OLE-интерфейсы, ведущие свое происхождение от DDE (Dynamic Data Exchange) — динамического обмена данными, позволяющего единственным способом в Windows осуществлять обмен информацией между процессами.

По сути своей COM является простым объектным расширением OLE-технологии, когда в локальной модели средств автоматизации OLE команды от клиента проходят через модуль-инициализатор (proxy), а сообщение — в модуль-переходник (stub), где оно разбирается, анализируется и откуда посылается команда на сервер, как показано на рис 1.1.

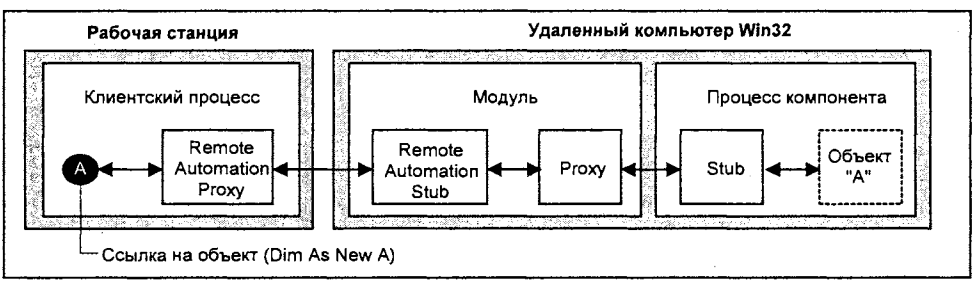


Рис. 1.1. Схема работы OLE Automation

Более наглядно эта же схема, но уже в спецификации COM, представлена на рис. 1.2.

Модель COM значительно проще с точки зрения программистов и ее гораздо удобнее использовать в небольших локальных сетях.

В дистанционной модели, которая теперь вместо COM получила название DCOM, инициализатор и переходник стали совершенно другими, теперь в них используются процедуры дистанционного вызова (RPC) для передачи запросов клиента по сети к модулю Remote Server Process, функционирующему на сервере. Клиентская часть DCOM, аналогичная прежнему меха-

низму Automation Manager, передает данные клиента указанному OLE-серверу и ответные данные по сети к программе-клиенту, как показано на рис. 1.3.

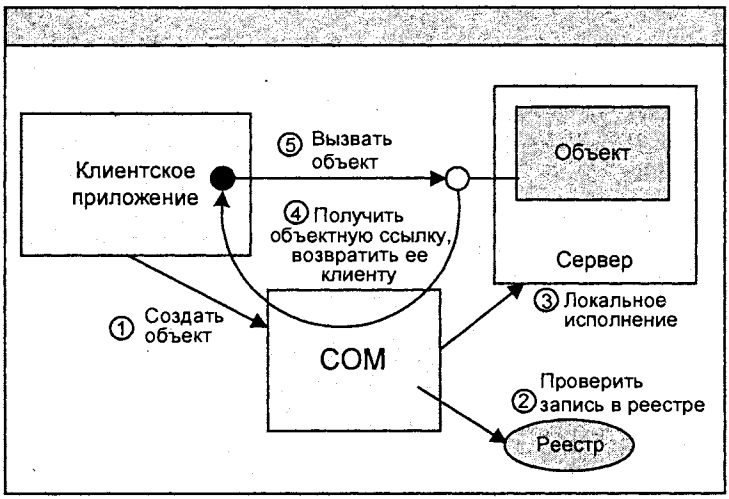


Рис. 1.2. Локальная схема взаимодействия объектов COM

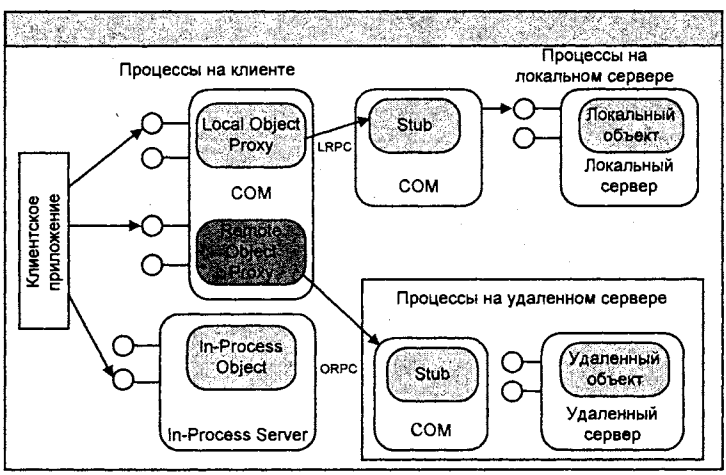


Рис. 1.3. Схема взаимодействия для локального и удаленного вариантов

Как можно видеть из приведенных рисунков, замечательным свойством DCOM является его прозрачность как для пользователя, так и для программиста. Можно создать и запустить на своей машине сервер OLE, а потом приспособить его для работы в дистанционном режиме простым перенесением OLE-сервера на удаленный ПК, на котором работает модуль-адми-

нистратор автоматизированного управления DCOM. Затем нужно просто зарегистрировать новое местоположение OLE-сервера с помощью обычной утилиты (например, из арсенала средств Visual Basic). Программисту не приходится менять ни одной строки текста программы, чтобы использовать разработанный OLE-сервер на удалении. Сейчас почти в любой RAD-системе имеются шаблоны для создания OLE-серверов, методы и объекты которых доступны при работе в дистанционном режиме. И, поскольку регистрацию удаленного сервера легко включить в процедуру установки программ-клиента, не приходится предпринимать каких-либо дополнительных действий, чтобы воспользоваться новыми возможностями.

Работа серверов OLE в дистанционном режиме имеет ряд существенных преимуществ, в том числе упрощенное администрирование и обслуживание, возможность его многократного использования любой программой-клиентом OLE, многоуровневая система защиты доступа и достоверности данных. Но самое главное преимущество — это способность удаленных OLE-серверов выполнять роль серверов приложений, т. е. своеобразных исполнителей алгоритмов (в новой терминологии — бизнес-правил) для доступа к данным в распределенных прикладных системах.

Реализация этой возможности представляет собой ключевой момент в создании трехуровневой архитектуры деловых приложений Microsoft, в которой часть пользовательского интерфейса размещена на рабочей станции клиента, бизнес-логика — на сервере среднего уровня, а сверхмощные средства обработки данных — на SQL-сервере базы данных (рис. 1.4).

Например, трехуровневая модель совершенно не нужна (и не обязательно пытаться делить программу на части), если она не решает общую проблему эффективности.

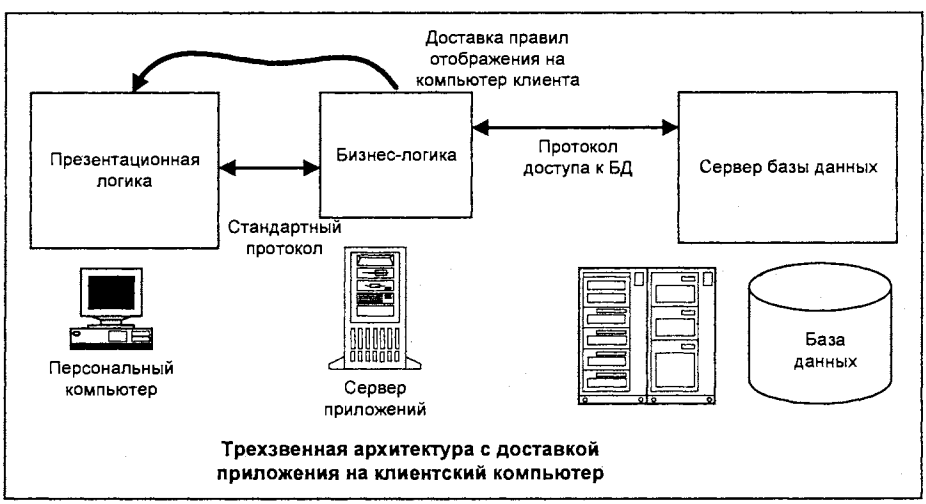


Рис. 1.4. Трехзвенная архитектура бизнес-приложений

Однако в фирменной документации Microsoft, посвященной OLE-объектам утверждается: "Чаще всего размещение сервер-объектов доступа к данным на той машине, где находится база данных, может существенно сократить загрузку сети и увеличить общую скорость передачи данных". Имеются примеры реализации трехзвенных клиент-серверных приложений, где результаты превосходят все ожидания. Производительность сети практически перестает влиять на время обработки транзакций, и своеобразное распиливание монолитных программных блоков на мобильные OLE-сегменты приводит к значительному повышению эффективности работы приложения в целом. Более того, трафик в сети существенно уменьшается и становится более предсказуемым.

Самый приемлемый, и наиболее простой, эффективный подход по мнению большинства создателей территориально-распределенных приложений — это создание трехзвенной архитектуры. В этой модели есть пользователь-клиент и сервер базы данных, а также средний уровень, который действует как "деловой" или интеллектуальный сервер приложений. Вы можете затем упростить прикладную программу пользователя, перемещая деловую логику вашей системы на сервер приложений, который мог бы находиться на том же самом компьютере, что и сервер базы данных (или на другом компьютере). Часто это дает эффект ускорения разработки и понижения дистрибутивных издержек для обслуживания "толстых" клиентов. Сервер приложений осуществляет запросы к серверу базы данных по требованиям клиентов и реализует бизнес-логику, что не только упрощает разработку и сопровождение клиентов, но также означает, что вы должны модифицировать только одну прикладную программу при изменении деловой логики реализации бизнес-процессов.

Чтобы лучше понять возможности распределенной архитектуры, рассмотрим пример из реальной жизни. Предположим, имеется большая транспортная компания M-Trans, которая обеспечивает своих клиентов специальной программой для отслеживания пути прохождения груза. Когда вы отправляете груз из Москвы в Дели, вам дается специальный транспортный номер. Затем, если вы захотите узнать, что случилось с вашим грузом на пути следования, вы просто загружаете программу (назовем ее MTrack) и вводите свой транспортный номер. Диалоговый модуль связывается с центральной универсальной ЭВМ компании и отыскивает информацию относительно текущего местоположения и состояния вашего груза. Например, вы могли бы узнать, что груз находится в настоящее время в узловом аэропорту отгрузки или что он был уже получен заказчиком. Электронная подпись заказчика, естественно, должна свидетельствовать об этом отрадном факте.

Пока все звучит, подобно банальной истории о типовом обслуживании клиентов на сервере. Однако проблема заключается в том, что программа сильно зависит от типа присвоенного транспортного номера. Различные типы чисел обозначают различные типы услуг, приобретаемые пользователем

(например, возможность удостовериться в подписи заказчика товара, подлинности упаковки и т. д.). Чтобы сделать MTrack настолько дружественной насколько это возможно, разработчики посчитали, что пользователь не должен нуждаться в интерпретации типа номера, который он получил. Было бы логично возложить данную процедуру на центральную ЭВМ, но в отношении нее у руководства фирмы существовала специальная политика и, кроме того, она была слишком загружена другими задачами. Ничего не оставалось, как вернуться к идее относительно выполнения этой простой логики на стороне пользователя. Впрочем, против такого решения возражала группа сопровождения ПО, мотивируя тем, что, поскольку, возможно, будут разрабатываться новые типы услуг, то программное обеспечение не будет распознавать эти новые услуги и должно время от времени модифицироваться. Так как программное обеспечение в данной транспортной фирме используется более чем 50 000 заказчиками, это было бы настоящим кошмаром. Поэтому было покончено с идеей интерпретировать транспортный номер пользователя с помощью соответствующего диалогового окна прикладной программы.

Более изящное решение в данном случае могло бы быть в создании сервера приложений на базе Windows NT в центре данных компании. Этот сервер приложений мог бы реализовать всю деловую логику, чтобы распознавать различные типы транспортных номеров. Программа клиента просто бы связывалась с сервером приложений, а он уже интерпретировал бы номер клиента и осуществлял соответствующий запрос на центральную ЭВМ. Это действительно устранило бы проблему модификации, потому что только сервер приложений должен был бы модифицироваться, если бы типы предлагаемых услуг изменились.

Этот пример хорош тем, что показывает, как надо использовать сервер приложений. Лучшим вариантом всегда будет тот, когда на сервере выполняется программный модуль, который может часто модифицироваться и который должны вызывать в своей работе одновременно тысячи клиентов. Вызывается он дистанционно и выполняет полезную работу. Клиент и сам бы мог иметь на своей машине этот модуль и все для него, клиента, было бы гораздо проще. Но дело в том, что клиентов — 50 000, а модуль часто модифицируется!

Все остальные доводы в пользу трехзвенной архитектуры сводятся пока к тому, что она лучше, потому что она лучше, — мол на клиенте ничего не надо инсталлировать, кроме интерфейсов к удаленному программному серверу. Но при этом, однако, не видно никакой особой выгоды, если программное обеспечение, которое переносится на сервер приложений, редко модифицируется. Загрузка сети остается той же, а на сервер падает львиная доля проблем, связанных с массовым обслуживанием клиентов, управлением транзакциями и отслеживанием очередей.

Вообще настройка любой существующей программы для применения в распределенной сети включает несколько шагов.

Пример создания объектно-распределенного приложения с использованием DCOM

1. Создадим на Visual Basic две простейшие программы: Hello-клиент (helo_cli.exe) и Hello-сервер (helo_svr.exe). Программа Hello-клиент должна содержать простейшую форму с кнопкой "Поприветствуй меня" (cmdSayHi), при нажатии на которую будет вызываться метод, печатающий в другой форме сообщение, присылаемое с сервера.

```
Option Explicit
Dim objNew As Object

Private Sub cmdSayHi_Click()
  MsgBox objNew.SayHello
End Sub

Private Sub Form_Load()
  Set objNew = CreateObject("HelloProj.HelloClass")
End Sub

Private Sub Form_Unload(Cancel As Integer)
  Set objNew = Nothing
End Sub
```

И сам метод и сообщение находятся в программе-сервере, которая оформлена как класс HelloClass. Эта программа содержит пустую форму (frmHello), имеющую некоторые пояснения для клиента:

```
' Пояснение 1: Следующие два сегмента кода демонстрируют две
' возможности для актуализации объекта со стороны клиента. Первая
' возможность использует раннее связывание и требует, чтобы
' "HelloProj" был выбран из RAD-системы Visual Basic по меню
' Tools,References. Вторая возможность использует позднее связывание
' и требует присутствия дополнительного оператора для создания
' объекта.
' Раннее связывание, конечно, быстрее, но позднее связывание дает
' большую свободу и гибкость для проектировщика.

' Пояснение 2: Клиентская часть кода присутствует в этом проекте
' только для облегчения отладки. Его можно без изменений вставить
' в любую клиентскую программу и использовать в режиме разделения.
```

```
' Раннее связывание
' Dim objNew As New HelloProj.HelloClass
' MsgBox objNew.SayHello
' Set objNew = Nothing

' Позднее связывание
' Dim objNew As Object
' Set objNew = CreateObject("HelloProj.HelloClass")
' MsgBox objNew.SayHello
' Set objNew = Nothing
End Sub
```

Главной частью на сервере, конечно, является модуль класса (`HelloClass`):

```
Public Function SayHello() As String
    SayHello = "Привет друг!"
End Function

Private Sub Class_Terminate()
  Unload frmHello
End Sub
```

2. В результате компиляции программы helo_svr.exe создается класс `HelloProj.HelloClass` и ему присваивается уникальный идентификатор `CLSID` (`{88C203D9-CA54-11D2-A604-204C4F4F5020}`), который указывает на регистрируемую программу helo_svr.exe. Эта информация, при построении программы в среде VB, автоматически заносится в реестр Windows, что позволяет увидеть программу в виде COM-объекта при использовании любого средства, способного просматривать все зарегистрированные объекты на локальной машине. Кроме того, при компиляции класса в каталоге проекта создается библиотечный файл Helo_svr.tlb со специальной структурой, доступной для анализа содержащихся в классе методов и их свойств, например с помощью программы, входящей в состав VB6, OLE/COM Object Viewer.

Пока все происходит на локальной машине. Что нужно сделать, чтобы вызывать методы сервера непосредственно из клиента, если сам сервер, т. е. разработанную нами программу helo_svr.exe, необходимо перенести на другую машину? Да почти ничего! Можно воспользоваться специальной утилитой Remote Automation Connection Manager, которая корректирует реестровые записи, или при желании самому скорректировать их вручную (категорически не рекомендуются средства Microsoft).

Что делает утилита Remote Automation Connection Manager?

Во-первых, она показывает в окне (рис. 1.5) все COM-объекты уже зарегистрированные на локальной машине (процесс регистрации происходит автоматически в момент построения и окончательной сборки COM-компонента). Обратите внимание: локальная машина — это, как правило, кли-

ентская машина и для каждого выбранного вами из списка COM-объектов в поле **Network Address** может стоять либо имя сервера, где должен выполняться COM-объект, либо вообще ничего (**None**), — т. е. этот объект должен быть зарегистрирован и выполняться на той же самой машине, где выполняется сама утилита. Попробуем перерегистрировать и нашу с вами программу helo_svr.exe, предвидя, что впоследствии мы перенесем ее на сервер с именем DUNAJ.

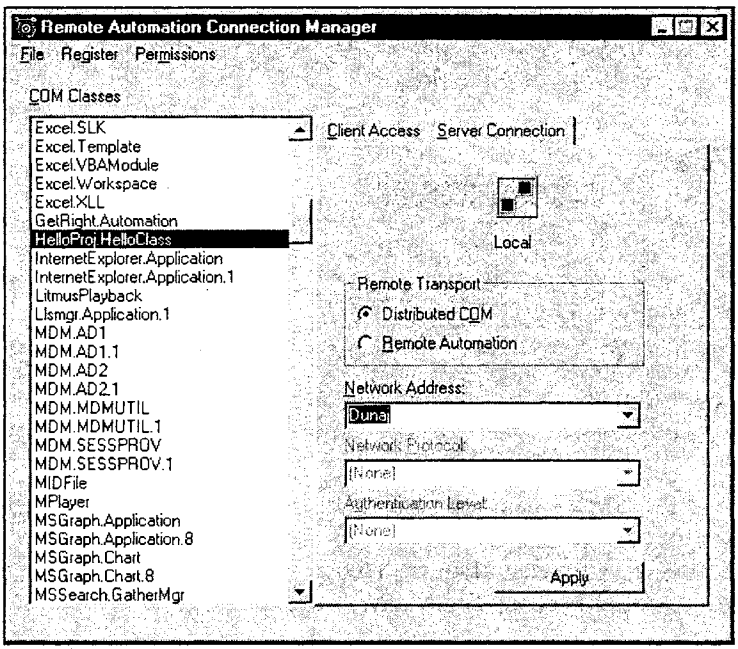


Рис. 1.5. Окно утилиты Remote Automation Connection Manager

Сразу после перерегистрации в базе данных реестра появится специальная запись с сетевым адресом для CLSID нашей программы, указывающая, что ее надо искать на сервере DUNAJ. Теперь надо просто перенести наш новоявленный COM-сервер (а попросту говоря, программу helo_svr.exe) на компьютер с именем DUNAJ и там запустить ее. Если компьютер DUNAJ доступен по сети, то можно на нашей локальной машине запустить программу-клиент helo_cli.exe и выполнить ее. Она установит соединение с сервером и вызовет желаемый метод. В окне формы вы получите ожидаемое сообщение (рис. 1.6).

На самом деле сообщение "Привет друг!" выдается сервером, хотя и появляется в окне клиента совсем на другом компьютере. Вы можете легко убедиться в этом, что-нибудь изменив в сообщении и перетранслировав программу на сервере. В последнем случае, ничего не меняя на стороне клиен-

та, вы увидите совсем другое сообщение. Все очень просто. Пользователю почти ничего не надо знать о том, как функционируют распределенные приложения. С его точки зрения ничего не меняется. Компонентная объектная модель COM плавно переходит в DCOM. У себя на компьютере надо только правильно зарегистрировать программу, методы которой вы хотите использовать. Вот и все. Остается открытым вопрос — нужно ли регистрировать программу, выполняющую роль сервера сообщений на компьютере DUNAJ? Надо, если вы не хотите каждый раз принудительно запускать ее. Кстати, технологии Microsoft допускают удаленную регистрацию. Кому-то это может показаться удобным, другие, наоборот, справедливо критикуют Microsoft за снижение требований безопасности.

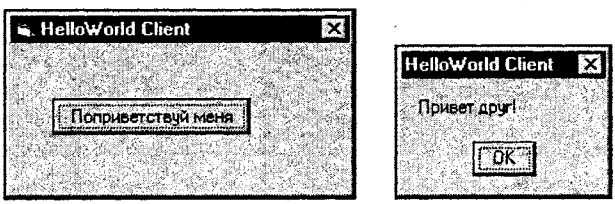


Рис. 1.6. Окно формы клиента и форма, присылаемая с сервера

В DCOM применяется динамический способ создания и уничтожения объектов, — пока объект востребован, он должен находиться в памяти, оставаться активным.

Таким образом, если у DCOM-серверного приложения есть активные клиенты, оно должно оставаться в памяти даже в те периоды времени, когда эти клиенты бездействуют. Справедливости ради, следует сказать, что в CORBA ресурсы используются гораздо более оптимально.

Обработка ненадежных соединений в DCOM организована гораздо хуже, чем в CORBA. Так, если соединение обрывается, сервер-приложение может решить выйти из памяти, но если клиент не завершил свою работу, то при восстановлении соединения он может просто "зависнуть". Microsoft декларировала, что в COM 3 будет предусмотрена обработка такого рода ошибок. Влияние компании Microsoft на рынок программных средств так велико, что если это не сделает сама Microsoft, то за нее это сделают другие компании. Именно поэтому в отношении не только DCOM, но и любой другой программной технологии Microsoft можно уверенно утверждать — они непоколебимы!

Главным недостатком CORBA является то обстоятельство, что имеется множество различных несовместимых реализаций CORBA различных производителей. DCOM — это средство для обслуживания компонентов ActiveX, которые обычно реализуют код для платформ Intel ПК. Трудно представить использование CORBA без технологий Java и JavaBeans фирмы Sun. Идея переносимых байт-кодовых модулей гораздо более привлекательна, чем идея

появления многоплатформенного DCOM. К сожалению, Java-технологии не слишком распространены в России, где уже долгое время усиленно размножается монокультура Wintel (операционная система Windows + компьютеры на платформе Intel).

Что подразумевается под технологиями ActiveX

Взгляд на некоторые особенности технологии

ActiveX служит одной единственной цели: обеспечивать функционирование программных компонентов внутри составных программных контейнеров. Эти контейнеры включают Web-браузеры и другие средства просмотра документов. ActiveX — технология Microsoft, предназначенная для написания сетевых приложений. Поскольку самым динамично развивающимся направлением в компьютерной индустрии является Интернет, именно здесь наиболее естественно могут найти свое место программы, написанные с использованием технологии ActiveX. Не случайно в последнее время понятия ActiveX и Интернета часто встречаются рядом. В то же время технология ActiveX имеет значительно более универсальную область применения.

Стандарт ActiveX позволяет программным компонентам взаимодействовать друг с другом по сети независимо от языка программирования, на котором они написаны. С помощью ActiveX можно "оживить" страницы Web эффектами мультимедиа, интерактивными объектами или сложными приложениями. ActiveX обеспечивает некие связные средства, с помощью которых отдельные программные компоненты на разных компьютерах "склеиваются" в единую распределенную систему.

ActiveX включает в себя клиентскую и серверную части, а также библиотеки для разработчика.

Управляющие элементы ActiveX (ActiveX Controls)

Что же это такое на самом деле? Наиболее простой ответ — это новое название для "управляющих элементов OCX" или даже для еще прежде них существовавших VBX. Любой готовый или созданный вами управляющий элемент OLE — это уже ActiveX-элемент, который может использоваться в программах. Однако подобные OLE-элементы во всем их бесчисленном многообразии вряд ли устроят разработчиков для Web.

Типичный недостаток существующих OLE-элементов — их значительные размеры. Это обусловлено сложностью структуры OLE-интерфейсов, а так-

же тем фактом, что при подготовке в Microsoft библиотек, используемых для генерации управляющих элементов, размеры их не оптимизировались. Если в системе пользователя какой-то из этих элементов отсутствует, приходится загружать его через Интернет, следовательно, размер управляющих элементов Web-страниц должен быть как можно меньше. С технической точки зрения ActiveX-элемент — это некоторый COM-объект, через основной OLE-интерфейс которого, IUnknown, организуется доступ к остальным интерфейсам данного объекта.

По существу с помощью ActiveX-элементов программист создает высокоуровневый, пригодный для многократного использования объект с некоторой полезной функцией. Затем этот элемент может быть передан (или продан) другому программисту, которому пригодится как некий "строительный блок". Большинство программных инструментальных систем, таких как Delphi, Visual Basic, Visual C++ поддерживают средства взаимодействия с ActiveX-элементами. В роли ActiveX-элементов может быть все что угодно — от обычной кнопки до полнофункциональной электронной таблицы. В продаже имеются тысячи таких элементов от разных поставщиков. Их богатые функциональные возможности и многообразие — отдельное, наиболее важное достоинство платформы ActiveX.

Вы можете спросить, а какое отношение это имеет к Интернету? Ответ в интерпретации Microsoft будет звучать так: самое прямое и непосредственное. По своей сути платформа ActiveX — это адаптация существующих технологий Microsoft применительно к Web. По заявлениям Microsoft управляющие элементы ActiveX работают быстрее, чем Java-апплеты. Сложные функции можно добавлять щелчком мыши. Главное, однако, не в том, что ActiveX, как и Java-апплеты, способны оживить Web-страницы. Куда важнее тот факт, что управляющие элементы ActiveX позволяют посетителям Web-узла выполнять сложные операции, получать нужную информацию из баз данных и от приложений, работающих на других серверах или даже на других Web-узлах.

Поскольку ActiveX-файл является 32-разрядным родственником VBX, он может использоваться и в обычном Visual Basic, и в языке сценариев VBScript. В табл. 1.1 перечислены основные библиотечные типы, используемые в Windows.

Таблица 1.1. *Библиотечные типы, используемые в Windows*

Управляющий элемент	Расширение	Функция
Динамически подключаемой библиотеки (Dynamic link library)	.dll	Позволяет пользователям получать доступ к функциям, подпрограммам и ресурсам из других приложений

Таблица 1.1 *(окончание)*

Управляющий элемент	Расширение	Функция
Visual Basic	.vbx	Обеспечивает тот же пользовательский сервис, что и DLL. Может применяться в интегрированных средах разработки, таких как Visual Basic 3.0, MSVC++ 1.52 и Delphi 1.0
ActiveX	.ocx	Обеспечивает пользовательский сервис, такой же, как DLL, и тот же, что и VBX. В добавление, OCX являются более функциональными и лучше используют все доступные им ресурсы. Поддерживаются практически всеми известными RAD-системами

Создание и регистрация управляющих элементов ActiveX

Нет ничего более простого и, одновременно, наиболее поражающего воображение программистов со старой практикой C++, чем создание управляющих элементов ActiveX (так называемых ActiveX Controls) из современных инструментальных средств программирования, таких как Delphi или Visual Basic. Современным разработчикам вовсе нет никакой необходимости досконально понимать сложную структуру встроенных интерфейсов, порой они даже не знают, что такое интерфейс `IUnknown` и зачем нужны функции `AddRef`, `Release` и `QueryInterface`. Зачем им это? Теперь все за них проделывают встроенные в современные RAD-системы "мастера" и шаблоны. Порой программисты вовсе не задумываются, какую работу проделывают за них TCP/IP и RPC, когда части приложения распределяются сразу на несколько систем при помощи "мастеров развертывания и инсталляции". Впрочем, об этом позже. Что же касается управляющих элементов ActiveX, то они являются первоосновой компонентного проектирования и программирования. Именно поэтому разработчики RAD-систем предельно облегчили работу по созданию таких строительных компонентов, из которых они и сами состоят почти на 80 процентов. Как создать и зарегистрировать на локальной, либо удаленной машине ActiveX-элемент, выполняющий некоторую полезную работу и который только при помощи буксировки с помощью мыши можно перетащить на палитру компонентов в RAD-системах или встроить в любое приложение, не исключая и Web-страницы, да так, чтобы его можно было сразу использовать? Как сделать, чтобы этот OCX-элемент мог загружаться и выполнять свою работу из браузера? Предположим, вам очень часто приходится писать Интернет-приложения, в которых на Web-странице должно

появляться окно со списком всех доступных на этой машине шрифтов. Не будете же вы, в самом деле, постоянно включать функцию на каком-нибудь JavaScript в каждый HTML-файл. Гораздо легче написать управляющий ActiveX-элемент. Попробуем сделать это на Visual Basic 6. Тут даже и программировать-то ничего не нужно. Запустим VB, выберем шаблон для приложения с элементом управления ActiveX и создадим в пустой форме стандартный элемент `ListBox`, также с помощью специального ActiveX-компонента, находящегося на стандартной палитре компонентов Visual Basic (рис. 1.7).

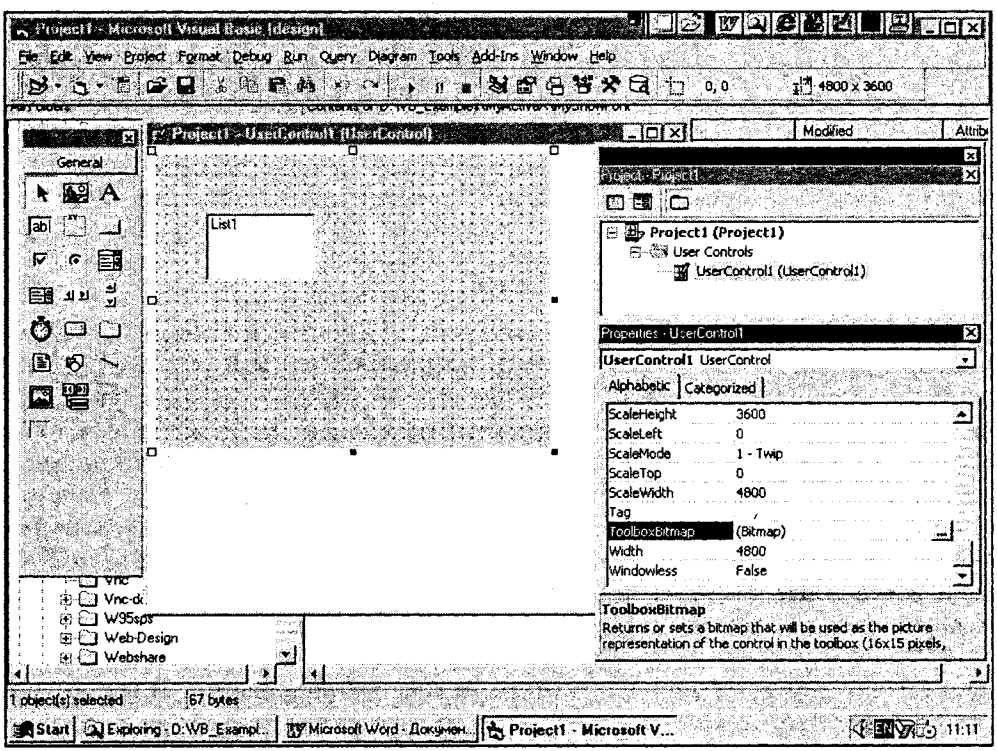


Рис. 1.7. Окно проекта для создания управляющих элементов ActiveX в VB 6

В свойствах **ToolboxBitmap** мы можем выбрать желаемую пиктограмму, в качестве которой создаваемый компонент будет присутствовать на палитрах компонентов в любой RAD-системе, в которой вы пожелаете его иметь. Далее нужно связать с этим компонентом какой-либо код. В нашем случае код должен быть самым примитивным — заполнение окошечка **ListBox** списком присутствующих на машине шрифтов. Программировать ничего не будем, возьмем стандартный код из примеров справочной системы VB 6 и "зальем" его в окно **Module1**, который предварительно создадим по команде **Project→ Add Module**. Ниже приведен этот код.

```
'Font enumeration types
Public Const LF_FACESIZE = 32
Public Const LF_FULLFACESIZE = 64

Type LOGFONT
        lfHeight As Long
        lfWidth As Long
        lfEscapement As Long
        lfOrientation As Long
        lfWeight As Long
        lfItalic As Byte
        lfUnderline As Byte
        lfStrikeOut As Byte
        lfCharSet As Byte
        lfOutPrecision As Byte
        lfClipPrecision As Byte
        lfQuality As Byte
        lfPitchAndFamily As Byte
        lfFaceName(LF_FACESIZE) As Byte
End Type

Type NEWTEXTMETRIC
        tmHeight As Long
        tmAscent As Long
        tmDescent As Long
        tmInternalLeading As Long
        tmExternalLeading As Long
        tmAveCharWidth As Long
        tmMaxCharWidth As Long
        tmWeight As Long
        tmOverhang As Long
        tmDigitizedAspectX As Long
        tmDigitizedAspectY As Long
        tmFirstChar As Byte
        tmLastChar As Byte
        tmDefaultChar As Byte
        tmBreakChar As Byte
        tmItalic As Byte
        tmUnderlined As Byte
        tmStruckOut As Byte
        tmPitchAndFamily As Byte
        tmCharSet As Byte
        ntmFlags As Long
        ntmSizeEM As Long
        ntmCellHeight As Long
        ntmAveWidth As Long
End Type
```

```
' ntmFlags field flags
Public Const NTM_REGULAR = &H40&
Public Const NTM_BOLD = &H20&
Public Const NTM_ITALIC = &H1&

'  tmPitchAndFamily flags
Public Const TMPF_FIXED_PITCH = &H1
Public Const TMPF_VECTOR = &H2
Public Const TMPF_DEVICE = &H8
Public Const TMPF_TRUETYPE = &H4

Public Const ELF_VERSION = 0
Public Const ELF_CULTURE_LATIN = 0

'  EnumFonts Masks
Public Const RASTER_FONTTYPE = &H1
Public Const DEVICE_FONTTYPE = &H2
Public Const TRUETYPE_FONTTYPE = &H4

Declare Function EnumFontFamilies Lib "gdi32" Alias _
    "EnumFontFamiliesA" _
    (ByVal hDC As Long, ByVal lpszFamily As String, _
    ByVal lpEnumFontFamProc As Long, LParam As Any) As Long
Declare Function GetDC Lib "user32" (ByVal hWnd As Long) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hWnd As Long, _
    ByVal hDC As Long) As Long

Function EnumFontFamProc(lpNLF As LOGFONT, lpNTM As NEWTEXTMETRIC, _
    ByVal FontType As Long, LParam As ListBox) As Long
Dim FaceName As String
Dim FullName As String
    FaceName = StrConv(lpNLF.lfFaceName, vbUnicode)
    LParam.AddItem Left$(FaceName, InStr(FaceName, vbNullChar) - 1)
    EnumFontFamProc = 1
End Function

Sub FillListWithFonts(LB As ListBox)
Dim hDC As Long
    LB.Clear
    hDC = GetDC(LB.hWnd)
    EnumFontFamilies hDC, vbNullString, AddressOf EnumFontFamProc, LB
    ReleaseDC LB.hWnd, hDC
End Sub
```

Код интересен тем, что он работает с WinAPI 32 и использует его стандартные функции, что само по себе в VB очень и очень привлекательно, ибо наделяет его необыкновенной мощью при использовании стандартных биб-

лиотек Windows, таких как GDI32.DLL, USER32.DLL, но нам сейчас гораздо интереснее другое — как упаковать это в переносимый, пригодный для повторного использования и применения где угодно, даже на Web-страницах Интернета новый ActiveX-элемент? Нет ничего проще. Продолжим работу в VB 6. После того как мы "залили" код в модуль, можно закрыть окно **UserControl**, и вы тотчас увидите выбранную вами пиктограмму в палитре инструментов. В окне **UserControl1** в параметре `Initialize` добавим строку `Module1.FillListWithFonts List1`. Стандартная процедура должна выглядеть так, как показано на рис. 1.8.

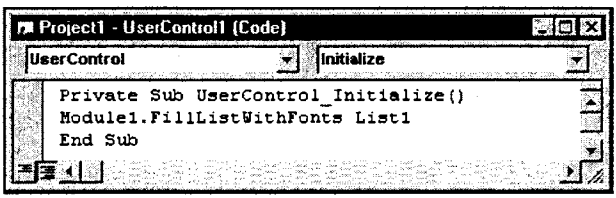


Рис. 1.8. Вызов функции модуля, реализующего функцию Windows API

Ну вот и все, можно поздравить вас — вы создали первый ActiveX-элемент. Выполнить его непосредственно из Бэйсика еще нельзя, хотя можно сохранить под любым именем (расширение .ocx файлу присвоится автоматически), но пока не будем делать этого, потому что такой шаг неизбежно связан с регистрацией нового элемента в системе (о ней мы расскажем позже). Попробуем использовать созданный нами управляющий элемент ActiveX. Через меню **File** добавим еще один проект к уже имеющемуся (**File→Add Project**). Выберем для нового проекта шаблон **Standard EXE**. Появится пустая форма. Втащим в эту форму из палитры компонентов наш ActiveX-элемент и придадим ему необходимые размеры. Теперь, при желании, можно сохранить группу проектов под любым именем по команде меню **File→ Sace Project Group As**, чтобы впоследствии продемонстрировать кому-нибудь ваши результаты или скорректировать проект. Хранить разрабатываемые ActiveX-элементы и формы их использующие в одной группе проектов очень удобно, потому что отпадает необходимость вручную включать их каждый раз в палитру компонентов из гигантского списка компонентов, уже зарегистрированных на вашей машине. Попробуем выполнить группу проектов по команде меню **Run** из среды Visual Basic. То, что вы увидите на экране, показано на рис. 1.9.

Как видите, запускается браузер по умолчанию, потому что используется файл с расширением .ocx. Он временно регистрируется в системе и исполняется. Разумеется, с данным списком шрифтов пока ничего нельзя сделать (только просмотреть их), но ведь это только пример! Вы можете разрабатывать для себя или с целью перепродажи куда более изощренные управляющие элементы со множеством свойств и методов, способных реагировать на различные события. Главное — это понять принципы создания таких элементов и их последующего использования.

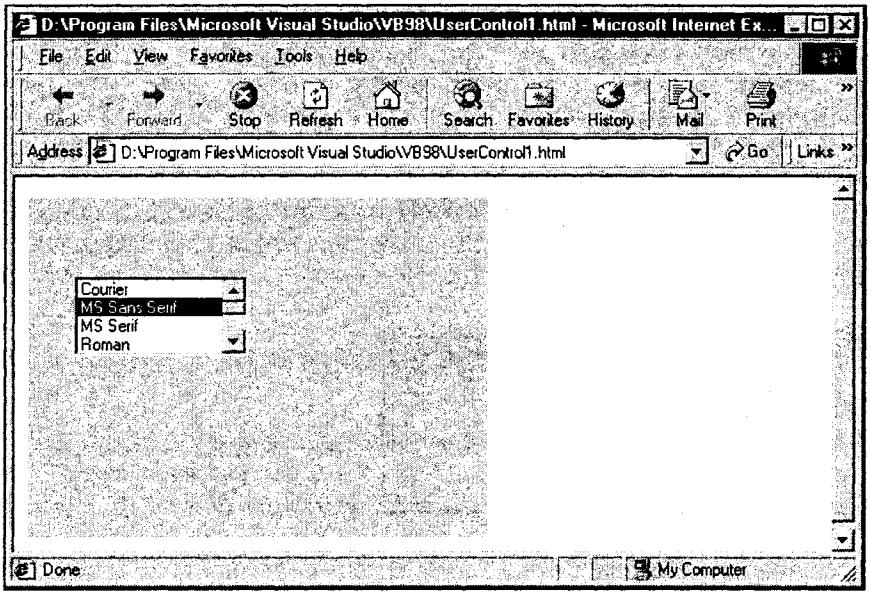


Рис. 1.9. Использование разработанного элемента ActiveX

Охарактеризуем вкратце базовые понятия и ключевые объекты, из которых строятся управляющие элементы ActiveX.

Контейнер и взаимодействие с ним

Управляющий элемент ActiveX не может существовать сам по себе, он всегда "живет" внутри своего контейнера и тесно взаимодействует с ним. Через свойства окружения (`ambient properties`) объект имеет доступ к информации о текущем состоянии своего "собственника". Контейнер поддерживает дополнительные методы, свойства и события, которые для разработчика выглядят как часть интерфейса элементов управления ActiveX.

Свойства окружения

Информация о состоянии контейнера доступна ActiveX-элементу через объект `AmbientProperties`, ссылка на который может быть получена через свойство `Ambient` объекта `UserControl`.

Например, свойство `UserControl.Ambient.BackColor` отображает текущее значение цвета фона контейнера и обычно используется при отображении ActiveX-элемента. Грамотно написанный ActiveX-элемент должен вести себя в соответствии с текущим состоянием контейнера. Сообщение `UserControl_AmbientChanged` извещает управляющий элемент ActiveX об изменении значения какого-либо свойства окружения контейнера.

Дополнительные свойства

Дополнительные свойства (`extender properties`) предоставляются контейнером, но внешне выглядят как часть интерфейса элементов управления ActiveX. Разработчик ActiveX-элемента имеет доступ к дополнительным свойствам через свойство `Extender` объекта `UserControl`. Спецификация элементов управления ActiveX требует, чтобы все контейнеры поддерживали следующие дополнительные свойства: `Name`, `Visible`, `Parent`, `Cancel`, `Default`. Для доступа к дополнительным свойствам всегда используется механизм позднего связывания (late-bound), т. к. на момент компиляции неизвестно с каким контейнером предстоит работать ActiveX-элементу.

Когда пользователь обращается к свойству (методу) `ActiveX Control`, то первым управление получает объект `Extender`. Если он не поддерживает это свойство (метод), то вызывается обработчик ActiveX Controls.

Объект *UserControl*

ActiveX-элемент, созданный на Visual Basic, всегда содержит (агрегирует) объект `UserControl`. Размещение и настройка свойств составляющих (constituent) ActiveX-элемента производится с помощью редактора свойств.

Объект `UserControl` имеет стандартный интерфейс. Вы можете написать свои обработчики для событий, генерируемых этим объектом, предоставить для использования или переопределить стандартные свойства. Таким же образом ваш ActiveX-элемент может экспортировать методы, свойства и события составляющих компонентов.

Ключевые события

В процессе своего существования ActiveX-элемент получает извещение о ряде событий, генерируемых объектом `UserControl`. Наиболее важными являются:

- **`Initialize`**. Самое первое событие, всегда происходит при создании ActiveX-элемента. В этот момент объект еще не имеет связи со своим контейнером.
- **`InitProperties`**. Это событие извещает о размещении нового ActiveX-элемента в форме. В момент его прихода встроенный объект уже имеет связь со своим контейнером. Как правило, обработчик события выполняет начальную инициализацию свойств ActiveX-элемента.
- **`ReadProperties`**. При загрузке формы (как в **DesignMode**, так и в **RunMode**) встроенные в нее элементы создаются заново и их свойства инициализируются значениями, сохраненными в файле с расширением .frm. Событие `ReadProperties` извещает ActiveX-элемент о необходимости выполнить инициализацию. В этот момент объект имеет связь со своим контейнером.

- **Resize**. Обработчик этого события отвечает за реакцию ActiveX-элемента на изменение его размеров.
- **Paint**. Обработчик этого события отвечает за отображение внешнего вида ActiveX-элемента. При установке свойства `UserControl.AutoRedraw=True` процесс визуализации происходит автоматически.
- **WriteProperties**. Извещает ActiveX-элемент о необходимости сохранить текущие значения его свойств. Генерируется в режиме **DesignMode** при сохранении формы-контейнера.
- **Terminate**. Извещает ActiveX-элемент об его уничтожении.

Последовательность событий четко определена и характеризует текущее состояние объекта. При размещении в форме нового ActiveX-элемента, загрузке формы, запуске проекта на исполнение (переходе из **DesignMode** в **RunMode**) все объекты создаются заново, свойства инициализируются сохраненными значениями (или значениями по умолчанию).

Создание программного интерфейса элемента управления ActiveX

Программный интерфейс ActiveX-элемента — это набор (public) методов, свойств и событий, посредством которых происходит взаимодействие объекта с его собственником, т. е. контейнером, или программой, которая использует этот контейнер.

По умолчанию интерфейс элемента управления ActiveX ограничивается набором методов, свойств и событий, предоставляемых контейнером. Задача разработчика заключается в добавлении новых (custom) свойств и методов, генерации событий и экспозировании, т. е выставлении на обозрение, собственных интерфейсов.

Свойства

Открытые свойства ActiveX-элемента необходимо реализовывать с помощью процедур `Property Get` и `Property Let`. Это требование связано с необходимостью извещать Visual Basic посредством вызова метода `UserControl.PropertyChanged` при изменении значения свойства пользователем. Выполнение этого требования необходимо по следующим причинам:

- если не вызвать метод `PropertyChanged`, то Visual Basic не передаст управление обработчику `UserControl_WriteProperties` для сохранения результатов редактирования в **DesignMode** и они будут утрачены;
- значение свойства может быть одновременно доступно в нескольких местах, например, в окне **Properties** и в диалоговом окне **Property Pages**. Вызов метода `PropertyChanged` синхронизирует отображаемую информацию;

- если для свойства созданы обе процедуры (`Property Get` и `Property Let`), то оно автоматически высвечивается в окне **Property**. Диалоговое окно **Property Attributes** меню **Tools** позволяет установить дополнительные атрибуты свойства.

Сохранение и восстановление состояния объекта

Каждый раз, когда форма загружается для редактирования или исполнения, все встроенные объекты создаются заново. Для сохранения свойств интерфейса элемента управления ActiveX между сеансами редактирования и для инициализации при запуске в **RunMode** служит механизм `Property Persistence`. Его реализация связана с тремя событиями: `InitProperties`, `ReadProperties` и `WriteProperties`, которые генерируются при размещении нового ActiveX-элемента в форме, считывании и сохранении состояния объекта, соответственно.

Обработчики `UserControl_WriteProperties` и `UserControl_ReadProperties` получают в качестве параметра объект `PropertyBag`. Он имеет два метода:

```
ReadProperty(Name As String, [DefaultValue]).
WriteProperty(Name As String, Value, [DefaultValue]).
```

Используя эти методы, разработчик ActiveX-элемента должен реализовать механизм сохранения и восстановления состояния объекта. В методе `WriteProperty` аргумент `DefaultValue` применяется для экономии места при записи. `WriteProperty` проверяет текущее значение сохраняемого свойства с `DefaultValue` и при равенстве запись не производит. Рекомендуется для каждого свойства определять константу, равную значению по умолчанию, и использовать ее при обработке событий `InitProperties`, `WriteProperties` и `ReadProperties`. В `UserControl_ReadProperties` необходимо предусмотреть проверку на корректность считываемых значений свойств.

Методы

Для реализации новых методов программного интерфейса элемента управления ActiveX достаточно добавить открытые (public) процедуры (subs) и/или функции (functions).

Для видимых в **RunMode** объектов необходимо, как минимум, реализовать метод `Refresh`. Обычно он просто вызывает `Refresh` объекта `UserControl`, при этом объект user-drawn получает сообщение `Paint` и производит свою прорисовку.

События

Генерация событий — важная особенность Visual Basic. События позволяют управляющему элементу ActiveX активно взаимодействовать со своим контейнером.

Механизм работы с событиями следующий (необходимо выполнить два шага):

- объявить событие с помощью выражения типа: `Public Event EventName [(arglist)]`;
- сгенерировать событие при помощи оператора `RaiseEvent`.

Рекомендуется, чтобы элемент управления ActiveX генерировал следующие стандартные события: `Click`, `DblClick`, `KeyDown`, `KeyPress`, `KeyUp`, `MouseDown`, `MouseMove` и `MouseUp`.

Представим, например, что вы хотите создать элемент управления в виде некоторой текстовой метки, которая меняет свое название и свойства, в зависимости от некоторого события — нажатия клавиши мыши. Пусть в нашем примере при первом нажатии мыши в поле метки она меняет свое название с **Метка** на **Старт**, а при повторном нажатии — на **Стоп**. Как должна выглядеть эта процедура в стандартном событии `Click` для объекта `Label1`, вставленном в `UserControl1`?

```
Private Sub Label1_Click()
a = Label1.Caption
If a = "Старт" Then
  a = "Стоп"
Else
  a = "Старт"
End If
Label1.Caption = a
End Sub
```

В стандартном окне разработки Visual Basic при выборе шаблона ActiveX-элемента это будет выглядеть так, как показано на рис. 1.10.

При исполнении данной программы в стандартном браузере это будет выглядеть так, как показано на рис. 1.11.

В этом примере вы можете видеть, что объект ActiveX работает и ведет себя точно так же, как любая VBScript-программа, встроенная в Web-страницу. Это — одно из больших преимуществ использования элементов управления ActiveX в Интернете. Широкий спектр выбора уже имеющихся в наличии и заказных ActiveX-элементов в конечном счете может привести к тому, что процесс программирования станет весьма легким делом: всего-то нужно будет только найти такой элемент управления, в котором разработчик нуждается для выполнения конкретной специфической цели.

Скажем еще несколько слов о регистрации компонентов ActiveX. Обычно Бэйсик самостоятельно регистрирует все ActiveX-компоненты в момент их создания. Регистрация заключается в присвоении ActiveX-компоненту специального уникального идентификатора CLSID и занесении в реестровую

базу Windows Registry нескольких записей, характеризующих свойства компонента и его местонахождение. Мы уже немного говорили о регистрации программ в разделах, посвященных COM и DCOM. Что касается дистрибуции и распространении разработанных ActiveX-компонентов по локальной сети или Интернет, то это отдельная тема и о ней скажем ниже.

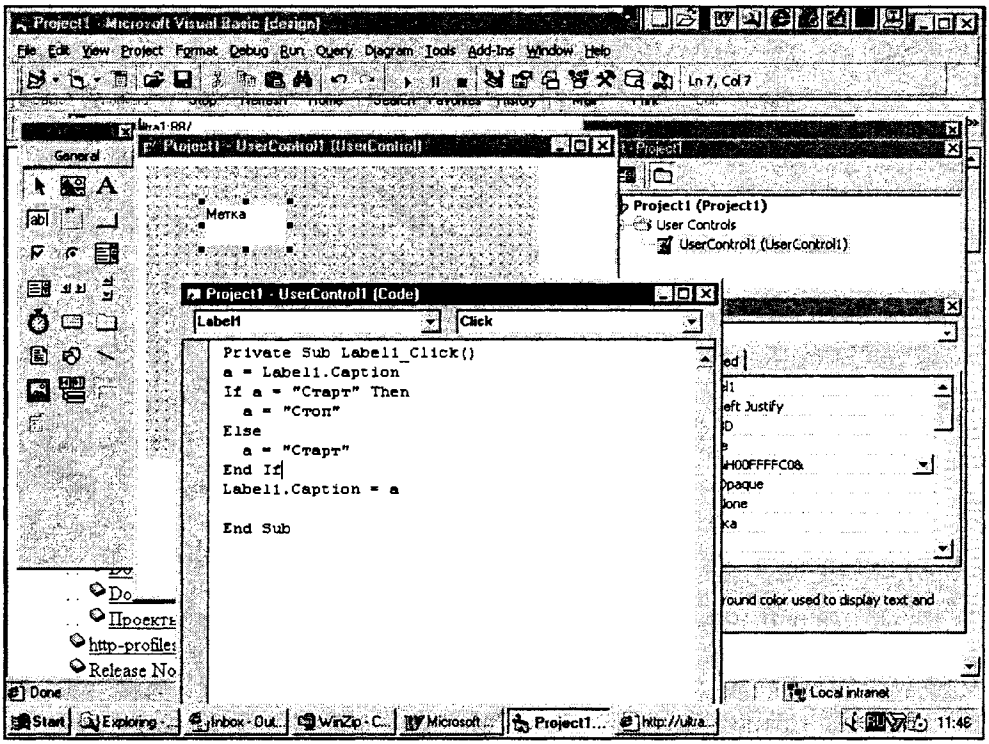


Рис. 1.10. Разработка элемента управления ActiveX в Visual Basic

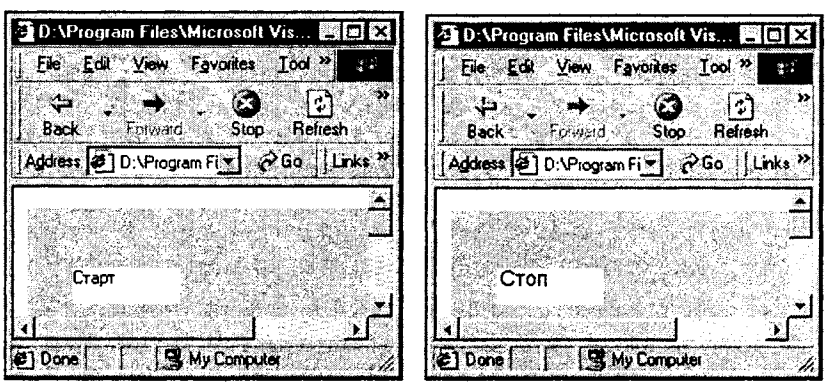


Рис. 1.11. Использование готового элемента управления ActiveX в браузере

Размещение управляющих элементов ActiveX на Web-страницах

Управляющие элементы ActiveX подключаются к HTML-документам с помощью тега <OBJECT>, который предложен в одном из рабочих стандартов консорциума W3 (см. **http://www.w3.org/pub/WWW/TR/WD-object.html**). Назначение этого тега (<EMBED> в Netscape Navigator) — интеграция в HTML чужеродных элементов. Вот полный синтаксис тега:

```
<OBJECT ALIGN=BASELINE|CENTER|LEFT|MIDDLE|RIGHT|TEXTBOTTOM|TEXTMIDDLE|TEXTTOP
  BORDER=число
  CLASSID=URL
  CODEBASE=URL
  CODETYPE=MIME-тип
  DATA=URL
  DECLARE
  HEIGHT=число
  HSPACE=число
  ID=идентификатор
  NAME=идентификатор
  SHAPES
  STANDBY=текст
  TYPE=MIME-тип
  USEMAP=URL
  VSPACE=число
  WIDTH=число
  >
 </OBJECT>
```

Некоторые из перечисленных атрибутов аналогичны одноименным атрибутам известного тега <IMG>, поэтому здесь мы их рассматривать не будем. Приведем описание атрибутов, специфических для тега <OBJECT>.

- **CLASSID**=URL

  С помощью этого атрибута указывается идентификатор класса вызываемого управляющего элемента. Каждый OCX-модуль и, следовательно, каждый управляющий элемент ActiveX обязан иметь свой уникальный идентификатор класса. Он обычно обозначается сокращением CLSID и представляет собой довольно длинную цепочку шестнадцатеричных цифр — например, "017C99A0-8637-11CF-A3A9-00A0C9034920".

  Но почему в синтаксисе этого атрибута стоит "URL"? Дело в том, что тег <OBJECT> может использоваться для подключения не только управляющих элементов ActiveX, но и других программных объектов, в том числе и таких, которые распознаются не по "идентификаторам классов", а как-то иначе. Чтобы браузер имел дело только с одним синтаксисом, идентифи-

катор CLSID задается по правилам URL: левая часть содержит слово "clsid", а правая — собственно идентификатор класса. Пример:

```
CLASSID="clsid:017C99A0-8637-11CF-A3A9-00A0C9034920"
```

- **CODEBASE**=URL

  В этом атрибуте указывается URL-адрес (на этот раз уже настоящий, с приставкой "http:" или "ftp:"). В этом атрибуте можно перечислить несколько URL-адресов — на тот случай, если один из серверов будет почему-либо недоступен.

  Этот же атрибут позволяет указывать для управляющих элементов ActiveX требуемый номер версии. Например, если написать

```
CODEBASE="http://activex.microsoft.com/controls/iexplorer/iestock.ocx#
Version=4,72,0,1171"
```

  то попытка получить данный модуль ActiveX из Интернета предпринимается не только в том случае, когда на компьютере пользователя его нет, но и тогда, когда управляющий элемент установлен, но имеет более старый номер версии, чем тот, что указан в атрибуте CODEBASE (в нашем примере требуется версия не старее 4.72.0.1171).

- **CODETYPE**=MIME-тип

  TYPE=MIME-тип

  Эти два необязательных атрибута позволяют указать типы (в терминах стандарта MIME) тех файлов, на которые ссылаются атрибуты CLASSID (атрибут CODETYPE) и DATA (атрибут TYPE). Для управляющих элементов ActiveX атрибут CODETYPE, если он присутствует, должен иметь значение "application/x-oleobject".

- **DATA**=URL

  С помощью этого атрибута указывается местонахождение файла данных, который может понадобиться для работы данному управляющему элементу. Для некоторых управляющих элементов ActiveX этот атрибут является обязательным. Кроме того, с помощью атрибута DATA можно включать прямо в HTML-файл в закодированном виде сам файл, содержащий управляющий элемент.

- **DECLARE**

  Обычно этот оператор пустой. Его можно использовать для объявления объекта, когда он не реализуется, а только является параметром другого объекта. При этом в память браузера записывается весь набор атрибутов объекта на тот случай, если их кто-то востребует.

- **ID**=*идентификатор*

  Нужен для организации взаимодействия с другими имеющими идентификатор объектами. Кроме того, может использоваться в схеме адресации URL (т. е. может стоять после символа "#" в URL-адресе).

Можно создать сколько угодно тегов <OBJECT> с одним и тем же значением CLASSID, но значения ID у них обязательно должны различаться.

- **NAME**=*идентификатор*

  Необязательный атрибут, который можно использовать при подготовке HTML-формы (для этого тег <OBJECT> должен находиться внутри пары тегов <FORM> ... </FORM>). Отправляемые данные будут содержать информацию, полученную браузером от объекта.

- **SHAPES**

  Этот пустой атрибут дает возможность в рамках окна занимаемого объектом на пространстве браузера выделять дополнительные подобласти и маркировать их как дополнительные URL-ссылки (своеобразные hotspots). Координаты этих регионов и URL-адреса ссылок для них указываются с помощью тегов <A> со специальными дополнительными атрибутами, которые должны располагаться между <OBJECT> и соответствующим ему </OBJECT>.

- **STANDBY**=*текст*

  В этом атрибуте можно указать текстовую надпись, которая будет видна в окне, отведенном объекту, пока сам объект будет загружаться и запускаться.

Помимо информации, передаваемой в атрибутах тега <OBJECT>, управляющим элементам ActiveX для работы требуется, как правило, набор каких-то дополнительных параметров.

Как и в случае оформления апплетов, о которых мы поговорим позже, эти параметры оформляются с помощью тегов <PARAM>, которые вставляются между <OBJECT> и соответствующим ему </OBJECT>. Каждый тег <PARAM> имеет пару атрибутов NAME и VALUE, с помощью которых задается одна пара "имя-значение", т. е. присваивается значение одному именованному параметру. Разумеется, список возможных параметров и набор допустимых значений никак не регламентирован — у каждого объекта они свои.

Например:

```
<OBJECT ID="Calendar" WIDTH=372 HEIGHT=279
 CLASSID="CLSID:8E27C92B-1264-101C-8A2F-040224009C02"
 CODEBASE="http://www.our_server.com">
    <PARAM NAME="_Version" VALUE="458752">
    <PARAM NAME="_ExtentX" VALUE="9843">
    <PARAM NAME="_ExtentY" VALUE="7373">
    <PARAM NAME="_StockProps" VALUE="1">
    <PARAM NAME="BackColor" VALUE="12161427">
    <PARAM NAME="Year" VALUE="1997">
    <PARAM NAME="Month" VALUE="2">
```

```
    <PARAM NAME="Day" VALUE="23">
    <PARAM NAME="GridCellEffect" VALUE="2">
    <PARAM NAME="GridFontColor" VALUE="65535">
</OBJECT>
```

Само название тега <OBJECT>, говорит нам о том, что при его использовании мы можем воспользоваться всеми преимуществами объектно-ориентированного программирования. Созданный таким способом объект имеет *свойства* (properties), *методы* (methods) и *события* (events), набор которых свой у каждого управляющего элемента ActiveX. Любой объект на Web-странице позволяет напрямую обращаться к его свойствам и методам из других объектов — не только из дополнительных управляющих элементов ActiveX, но и из встроенных в HTML-документ сценариев на любом из поддерживаемых языков (VBScript и JavaScript).

Переменные свойства объекта всегда совпадают (и по значению, и по обозначению) с теми параметрами, которые передаются ему с помощью тегов <PARAM>. *Методы объекта* — это функции, вызов которых позволяет выполнить какие-то действия, специфические для данного класса объектов. Доступ к свойствам и методам объекта в языке JavaScript осуществляется через идентификатор, присвоенный объекту атрибутом ID. Чтобы получить значение какого-то свойства объекта, достаточно написать

имя-объекта.имя-свойства

а чтобы вызвать принадлежащий объекту метод следует написать

имя-объекта.имя-метода (параметры метода)

Доступ к событиям объекта в теге <OBJECT> оформляется несколько по иному, чем в JavaScript. Нужно записать обработчик события в отдельной паре тегов <SCRIPT> ... </SCRIPT> и добавить к открывающему тегу следующую пару атрибутов:

FOR="*имя объекта*"
EVENT="*имя события*"

Имя объекта — это тот идентификатор, который был ему приписан с помощью атрибута ID. В принципе можно использовать события и гораздо более простым способом. Всем известен распространяемый повсеместно управляющий элемент ActiveX с названием Popup Menu.

Элемент управления Popup Menu

Элемент управления Popup Menu используется для создания выпадающего меню, появляющегося при нажатии пользователем на кнопку или картинку на Web-странице, связанных с этим управляющим элементом. Его довольно часто используют Web-мастера, чтобы дать возможность пользователям са-

мим выбирать из многочисленных ресурсов, красиво упакованных в удобные списки со скроллингом (с полосами прокрутки), что им нужно.

Элемент управления Popup Menu имеет только одно свойство `ItemCount`. Это свойство определяет количество элементов в списке. Параметр `Menuitem[]` используется, чтобы маркировать выбранные элементы. Вы можете использовать нижеприведенный скрипт, чтобы вызывать различные URL, которые предлагаются в списке меню. Имеются пять методов, связанных с элементом управления Popup Menu:

- `AboutBox` — предоставляет информацию об управляющем элементе;
- `Additem(`*`item, index`*`)` — добавляет к списку новую запись;
- `Clear` — очищает список меню;
- `PopUp(`*`x, y`*`)` — определяет позиции *x* и *y*;
- `RemoveItem(`*`index`*`)` — удаляет запись из списка.

Обрабатываемое событие для элемента управления Popup Menu — это нажатие кнопки мыши (`Click(`*`item`*`)`). Индекс записи из списка при этом передается в программу.

Приведем пример использования этого управляющего элемента:

```
<HTML>
<HEAD>
<TITLE>Test Page</TITLE>
</HEAD>
<BODY BGCOLOR=WHITE>
<H1>Popup Menu Control</H1>
<HR COLOR=NAVY>

<Table BORDER BGCOLOR=YELLOW></TR><TD>
<OBJECT ID="Label1" WIDTH=165 HEIGHT=44
 CLASSID="CLSID:99B42120-6EC7-11CF-A6C7-00AA00A47DD2">
    <PARAM NAME="_ExtentX" VALUE="4366">
    <PARAM NAME="_ExtentY" VALUE="1164">
    <PARAM NAME="Caption" VALUE="Popup Menu">
    <PARAM NAME="Angle" VALUE="0">
    <PARAM NAME="Alignment" VALUE="4">
    <PARAM NAME="Mode" VALUE="1">
    <PARAM NAME="FillStyle" VALUE="0">
    <PARAM NAME="FillStyle" VALUE="0">
    <PARAM NAME="ForeColor" VALUE="#000000">
    <PARAM NAME="BackColor" VALUE="#C0C0C0">
    <PARAM NAME="FontName" VALUE="Arial">
    <PARAM NAME="FontSize" VALUE="18">
    <PARAM NAME="FontItalic" VALUE="0">
    <PARAM NAME="FontBold" VALUE="1">
```

```
    <PARAM NAME="FontUnderline" VALUE="0">
    <PARAM NAME="FontStrikeout" VALUE="0">
    <PARAM NAME="TopPoints" VALUE="0">
    <PARAM NAME="BotPoints" VALUE="0">

</OBJECT>
</TR></TD></TABLE>
<OBJECT ID="Pop1" WIDTH=5 HEIGHT=5
 CLASSID="CLSID:7823A620-9DD9-11CF-A662-00AA00C066D2">
    <PARAM NAME="MenuItem[0]" VALUE="Домашняя страница Microsoft">
    <PARAM NAME="MenuItem[1]" VALUE="Поисковый сервер">
    <PARAM NAME="MenuItem[2]" VALUE="PCWeek Русская редакция">
</OBJECT>

<Script Language=VBScript>

Sub Label1_Click
    call Pop1.PopUp()
End Sub

Sub Pop1_Click(byVal X)

Select Case X
    Case 1
        top.Location="http://www.microsoft.com"
    Case 2
        top.Location="http://www.rambler.ru"
    Case 3
        top.Location="http://www.pcweek.ru"
End Select

End Sub
```

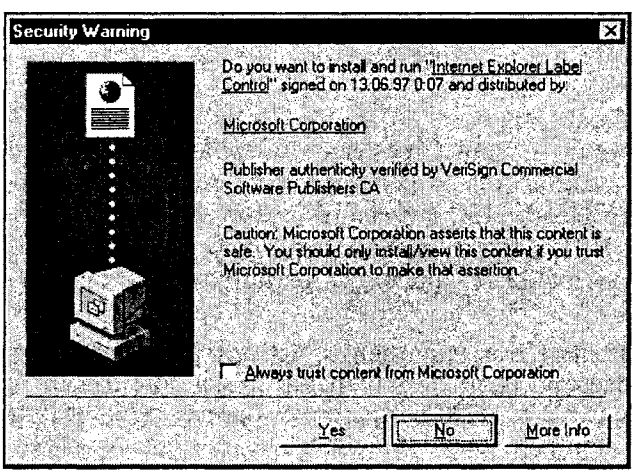


Рис. 1.12. Окно-предупреждение при копировании и регистрации элемента управления

Вы можете легко переписать этот сценарий и добавить в выпадающее меню адреса и ссылки, какие пожелаете.

Интересно, что если вы скопируете из книги этот кусок кода в какой-нибудь файл с расширением .html и запустите его из браузера, то отсутствующий у вас элемент управления ActiveX скачается по Сети с выдачей предупреждающего окна (рис. 1.12).

Инсталляция и распространение управляющих элементов ActiveX

Обычно системы для разработки управляющих элементов ActiveX, такие как Delphi, MS C++ или Visual Basic, уже содержат в себе средства для их автоматической регистрации и распространения. Например, Visual Basic включает целый набор подобных средств. Разработка компонентов на Visual Basic может быть проведена двумя способами:

- проект элемента ActiveX может содержать несколько модулей ActiveX (файлов с расширением .ctl). При этом управляющие элементы, имеющие свойство `UserControl.Public=True`, доступны вне проекта для всех приложений;
- закрытые (private) управляющие элементы ActiveX могут быть включены в проект любого типа, но они доступны только внутри своего приложения.

Если вы хотите, чтобы созданные вами управляющие элементы ActiveX были доступны другим разработчикам, следует позаботиться о том, чтобы вместе с ними поставлялись и необходимые библиотеки. Например, компоненты, созданные на Visual Basic, требуют при исполнении наличия динамических библиотек (VB run-time DLL). Приложение Setup Wizard выполняет всю работу для создания стандартной программы установки.

Процесс распространения ActiveX-компонентов может быть лицензирован.

Лицензирование

Visual Basic поддерживает механизм защиты разработанных вами компонентов от нелегального использования. Для включения этого механизма необходимо выбрать пункт **Require License Key** в диалоге установки свойств проекта. При этом будет создан VBL-файл с ключом (registry key). Кроме того, мастер (Wizard) автоматически добавит процедуру регистрации устанавливаемого компонента.

При попытке использования элементов управления ActiveX в процессе создания приложения будет проверено наличие лицензии на машине разработчика. При использовании ActiveX в HTML-страницах браузер клиента запрашивает у сервера лицензионный ключ и использует его для создания

объекта, при этом полученная лицензия в регистрационную базу (Registry) компьютера клиента не добавляется.

Любые ActiveX-элементы, которые нужно взять со страниц Web, должны быть сначала загружены на клиентские машины. Это является ответственностью браузера — копировать требуемые файлы и устанавливать компоненты только в том случае, если они еще не существуют. Чаще всего используются следующие варианты.

- **CAB-файлы.** Все разработанные управляющие элементы обычно упаковываются в стандартный архив Microsoft с расширением .cab. Гораздо проще иметь дело с одним файлом, который содержит множество других файлов. Кроме того, поскольку он в сжатом формате, то и быстрее загружается по сети. Браузер понимает формат этого файла и сам его распаковывает.
- **MSVBVM60.DLL.** Библиотека поддержки, требуемая для всех управляющих элементов, созданных в Visual Basic. Если этот файл не размещен на клиентской машине, процесс SETUP определяет, куда он должен быть выгружен.
- **Цифровые подписи.** Гарантируют безопасность загружаемого элемента управления, потому что любому ActiveX-компоненту, в отличие от Java-апплетов, в принципе, могут быть доступны все ресурсы операционной системы.
- **Совместимость версий.** Гарантирует, что клиентская машина имеет, по крайней мере, ту же самую версию загружаемого компонента. При наличии на машине клиента устаревшей версии компонента, браузер запрашивает загрузку более свежей версии.

Например, когда мы входим на сайт "Музеи Росии" (www.museum.ru) и выбираем панорамную экспозицию (**www.museum.ru/gmii/exp_4_1.htm**) для путешествия по залам Государственного музея изобразительных искусств им. А. С. Пушкина, то в результате с этой страницы, согласно размещенному на ней тегу

```
<OBJECT>
<script language="JavaScript">
<!--
//navigator.appName == "Microsoft Internet Explorer" &&
parseFloat(navigator.appVersion) >= 3.0
if (true)
document.write('<OBJECT CLASSID="clsid:928626A3-6B98-11CF-90B4-
00AA00A4011F" TYPE="application/x-oleobject" ID=Panorama
CODEBASE="MSSurVid.cab#Version=1,2,0,10" width=500 height=250>'+
'<PARAM NAME="Image" VALUE="img/pan_new1.svj"><PARAM NAME="SurroundRect"
VALUE="0,0,500,250"></OBJECT>')
else document.write('<FONT SIZE="+2" COLOR="MAROON">Only for MS Internet
Explorer v.3.0 or better!</FONT>')

//-->
</script>
```

к нам на компьютер после соответствующего предупреждения (рис. 1.13) загружается ActiveX-объект с возможностями управления изображением.

Этот объект скачивается по Интернету в том случае, если он на вашей машине еще не зарегистрирован, регистрируется в базе Registry под именем Surround Video Control Object и получает управление от браузера. Сам файл с именем MSSurVid.cab размещается, если вы, например, работаете в Windows NT в каталоге c:\winnt\Downloaded Program Files. Можно войти туда и посмотреть свойства этого файла (рис. 1.14).

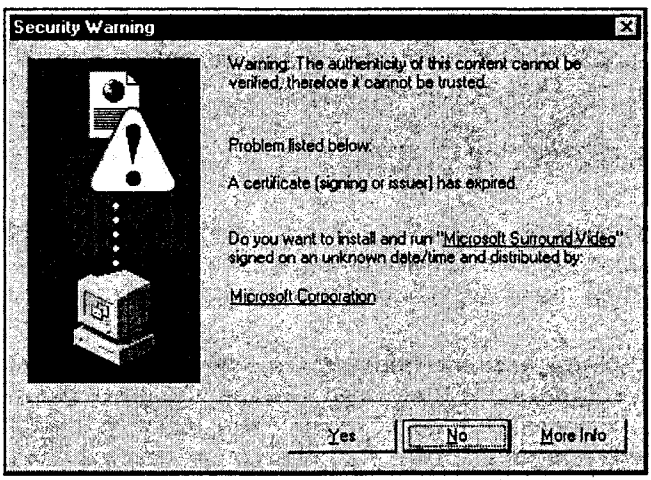


Рис. 1.13. Предупреждение о возможности "закачки" ActiveX-элемента

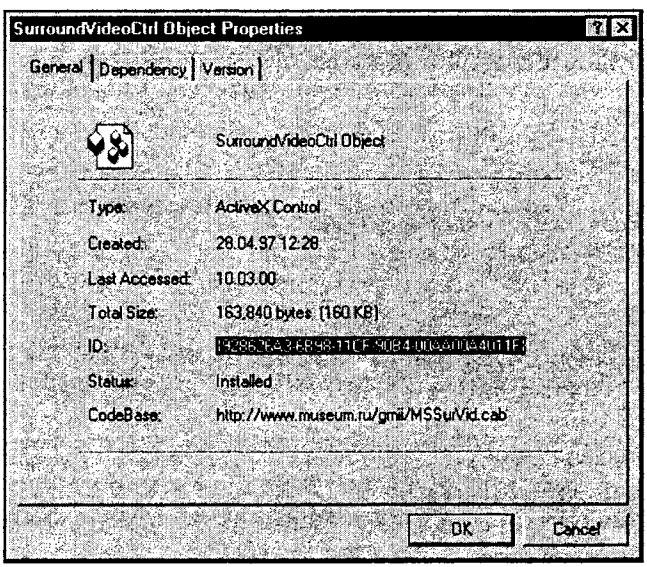


Рис. 1.14. Свойства скачанного по Интернету ActiveX-элемента

Как видно, файл представляет собой небольшой программный архив (на вкладке **Dependency** можно увидеть, что он включает в себя 3 элемента: CPSurVid.dll, MSSurVid.inf и MSSurVid.ocx) и имеет уникальный идентификатор, а также поле **CodeBase**, которое сообщает, откуда был закачан файл.

ActiveX-библиотека (ActiveX DLL)

Начинающие разработчики Visual Basic, Delphi или другой RAD-системы, впервые запустив графическую оболочку среды программирования и увидев в окне все многообразие типов создаваемых программ, среди которых им надо выбрать только один желаемый тип, зачастую теряются в догадках — какой шаблон для своей будущей программы выбрать, — потому что на панели выбора они видят картину, подобную показанной на рис. 1.15.

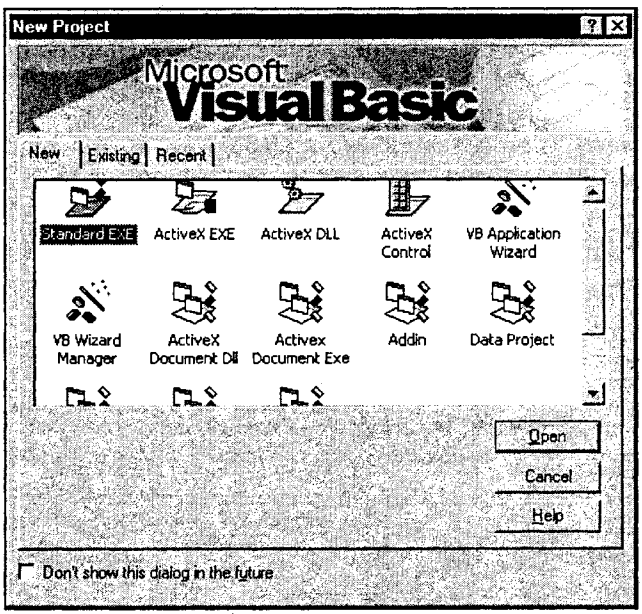


Рис. 1.15. Окно выбора типа программы в VB 6

Чем, например, отличается стандартный EXE-файл от ActiveX EXE-файла, а тот, в свою очередь от файла ActiveX DLL? Какая между ними разница? Если внимательно посмотреть на рисунок, то можно увидеть еще ActiveX Document Dll и Active Document Exe. Только одних типов ActiveX — 5 штук, в самом деле голова может пойти кругом! Однако это вовсе не недостаток, а достоинство современных средств разработки, темп развития которых определяет в настоящее время Сеть (имеется в виду Интернет), только Сеть и ничего больше. Программисту предлагается такое разнообразие типов создаваемых программ, только потому, что сама ткань Web и множество переплетающих ее объектов становятся все более и более разнообразными. А ведь

мы пока ведем речь только о технологиях Microsoft. Что будет дальше, когда мы с головой окунемся в еще более разнообразный мир технологий Java? Давайте сначала все-таки разберемся с тонкостями отличий программ и модулей ActiveX.

Ключевые понятия: in-process и out-of-process

Разница между файлами DLL и EXE по отношению к ActiveX всегда одна — это контекст, в котором они выполняются. Если ActiveX-компонент выполняется в адресном пространстве приложения, которое его использует, то это всегда DLL, или *in-process*. Естественно, что это самые быстрые и эффективные процессы, ибо им не требуется *маршаллинг*, т. е. выход за границы процесса, их породившего, и разрешение связанных с этим проблем. Спрашивается, а чем же тогда отличается библиотека ActiveX DLL от обычной DLL? Ответ — ничем не отличается, это такая же обычная библиотека DLL, функции которой можно вызывать из скомпонованных готовых программ без их перекомпоновки — гениальное изобретение Microsoft. Что касается *out-of-process*, то это, чаще всего, программы ActiveX EXE, которые выполняются на удаленных машинах (но не обязательно) и реализуют логику сервера приложений. Об этом мы расскажем чуть ниже, а пока на рис. 1.16 показана взаимосвязь различных типов ActiveX-компонентов, используемых в одном проекте.

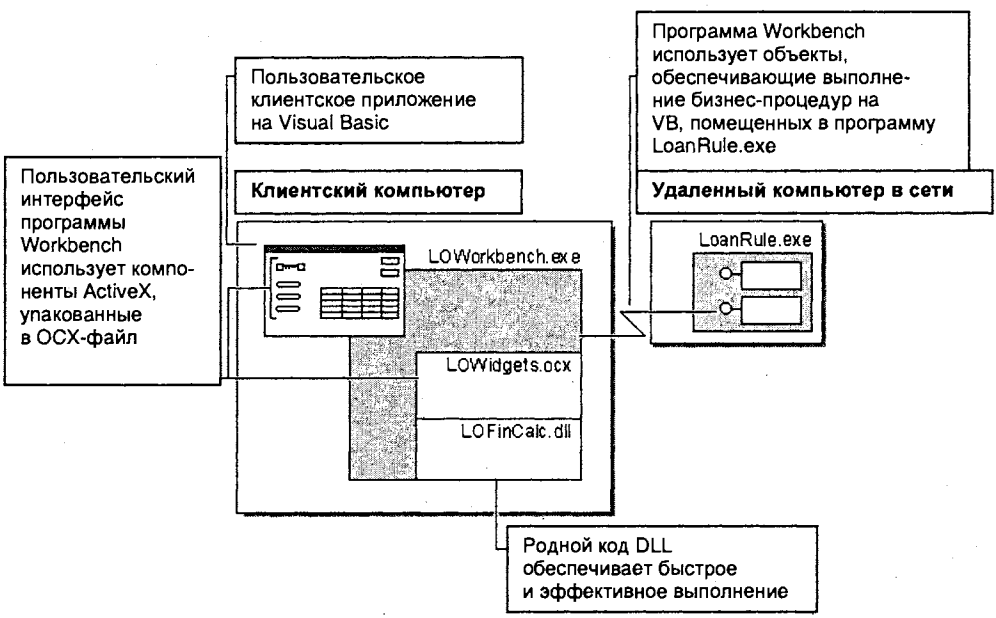


Рис. 1.16. Использование нескольких типов ActiveX в одном проекте

На этом рисунке можно видеть 4 типа различных программ, составляющих одно целое.

ActiveX DLL обычно используют для программирования часто применяемых и полезных функций, чтобы не включать их по несколько раз в каждый разрабатываемый модуль. Стоит лишь однажды написать для себя хорошую и полезную библиотеку DLL (на любом языке программирования) и вы уже во всех своих последующих проектах просто не сможете отказаться от этого чрезвычайно удобного метода, без которого сейчас даже и нельзя помыслить разработку крупных программных комплексов. Библиотеки ActiveX DLL могут располагаться на серверах и исполняться из браузера в контексте Web-сервера с передачей им необходимых параметров и вызовом определенных функций. При этом Web-страница, которая в конечном счете передается в браузер, уже может содержать элементы оформления — результаты работы DLL (например, счетчик посещений Web-сервера — наиболее типичный способ использования DLL). К сожалению, способы программирования с использованием DLL выходят за пределы настоящей книги, но тысячи примеров содержатся и в Интернете (*см. ссылки в конце главы и в фирменной документации по Visual C++, Visual Basic и Delphi*).

ActiveX-сервер (ActiveX EXE)

В самом начале этой главы уже приводился пример программы helo_svr.exe, которая может выполнять роль ActiveX-сервера и содержит в себе методы, доступные клиентам через DCOM дистанционно. Это типичный пример программы типа out-of-process. Схематично, в отношении некоторых условных программ (клиент и сервер) это может выглядеть так, как показано на рис. 1.17.

Как можно использовать ActiveX-серверы или ActiveX-контейнеры? Наиболее типичный случай — это публиковать и широко рекламировать свою объектную модель, чтобы к описанным в ней объектам, их методам и свойствам могли свободно получать доступ другие программы. Почти все известные программы и СУБД, которые вы уже инсталлировали на своем компьютере, являются ActiveX-контейнерами или ActiveX-серверами. Запустите из Microsoft Studio утилиту OLE/COM Object Viewer и вы увидите много интересного. С помощью различных программ-анализаторов или фирменных описаний можно узнать, какие конкретно методы доступны для вызова и использования. Если вы знаете их, то свободно сможете осуществлять различные манипуляции из ваших программ, например, над таблицами Excel и документами Word, потому что и Word, и Excel являются ActiveX-серверами, а их объектные модели широко известны почти любому разработчику. Более того, мне однажды понадобилось осуществить доступ из базы данных Lotus Notes к базе данных Oracle, запущенной совсем на другом компьютере. Поскольку обе базы данных управлялись Windows, использование технологии ActiveX оказалось как нельзя кстати.

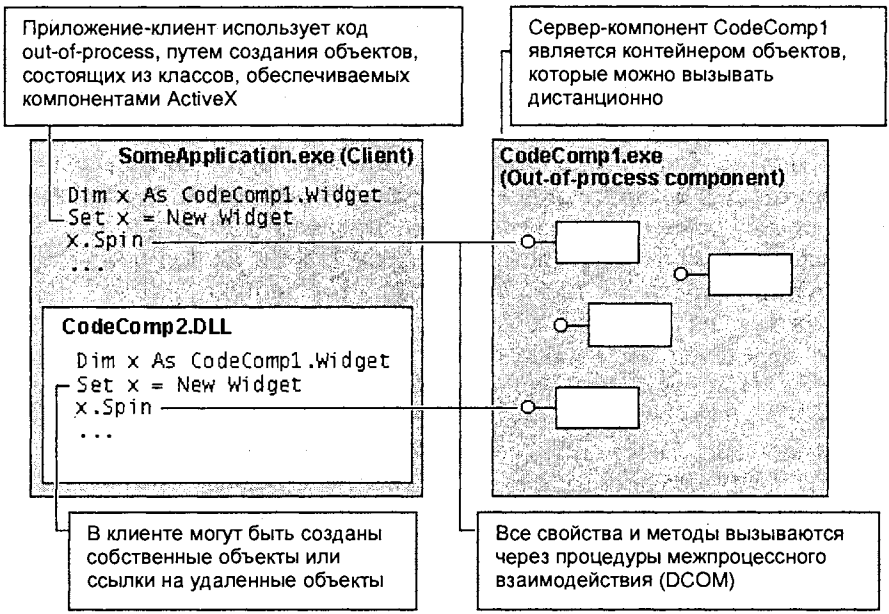


Рис. 1.17. Схема построения распределенных приложений по технологии ActiveX

Пример взаимодействия Oracle и Notes с использованием технологий ActiveX

Что такое Oracle COM Automation Feature

Oracle COM Automation Feature позволяет PL/SQL-разработчикам манипулировать COM-объектами через интерфейс OLE Automation Idispatch (рис. 1.18).

В PL/SQL-пакетах можно определять наборы интерфейсов (APIs) для инициации COM-объектов, получать или устанавливать их свойства и вызывать их методы. PL/SQL-разработчики могут вызывать эти APIs из PL/SQL-подпрограмм, сохраненных процедур или триггеров для манипуляции с COM-объектами. Доступ к объектам, где бы они не находились, осуществляется совершенно прозрачно: либо непосредственно в локальной СУБД, либо в удаленной посредством протокола DCOM (Distributed Component Object Model).

Таким образом у разработчиков приложений для баз данных Oracle появляется возможность улучшать их функциональность путем использования средств из арсенала Microsoft Word, Microsoft Excel и Microsoft PowerPoint, генерировать отчеты, используя Crystal Reports, получать и посылать почту в форматах почтовых серверов Lotus Notes, Microsoft Exchange или Oracle InterOffice.

Эти APIs выполнены в виде внешних процедур или динамически подключаемых библиотек (DLL).

Oracle8 версии 8.0.3 стал первой реализацией для бесшовного включения внешних процедур.

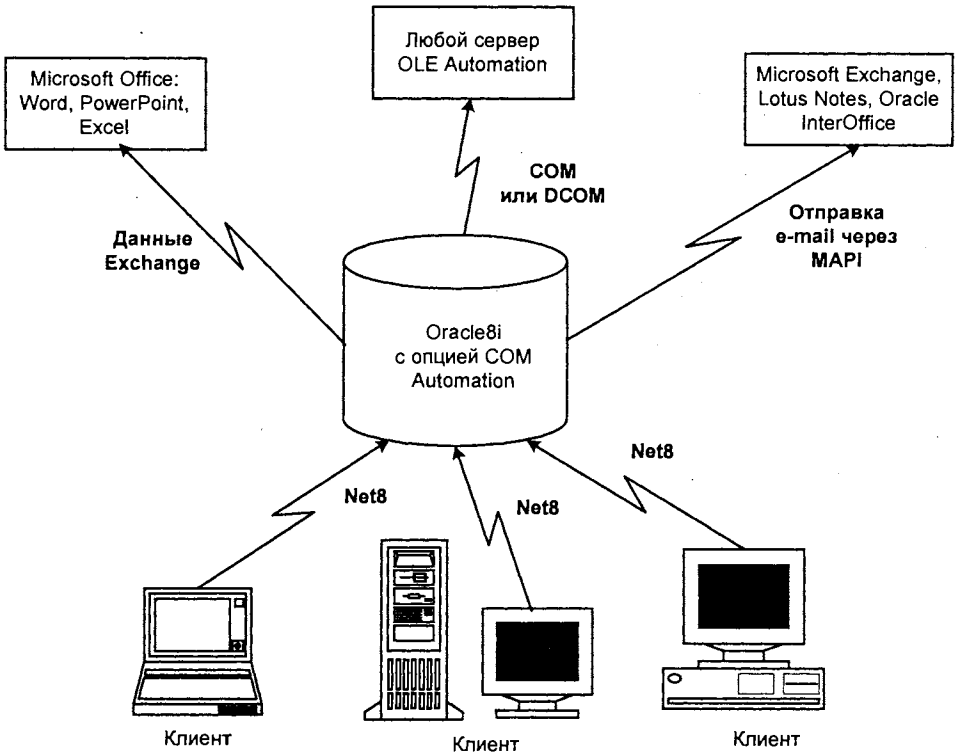


Рис. 1.18. Использование COM/DCOM в Oracle

Реализация интерфейса подключения внешних процедур OLE Automation External Procedure APIs

СУБД Oracle вызывает любую внешнюю OLE Automation APIs следующим способом:

1. PL/SQL-интерпретатор определяет местонахождение специальной библиотеки (Oracle COM Automation feature) (COM81.DLL).
2. PL/SQL-интерпретатор посылает сообщение, используя Net8 к Oracle-listener для запуска EXTPROC.EXE.
3. EXTPROC.EXE загружает DLL и выполняет внешнюю процедуру. Каждая внешняя API-процедура OLE Automation вызывает Win32 APIs, кото-

рый инициирует COM-объект, устанавливает или получает его свойства и вызывает его методы.

4. EXTPROC.EXE действует как часть Oracle Server (рис. 1.19).

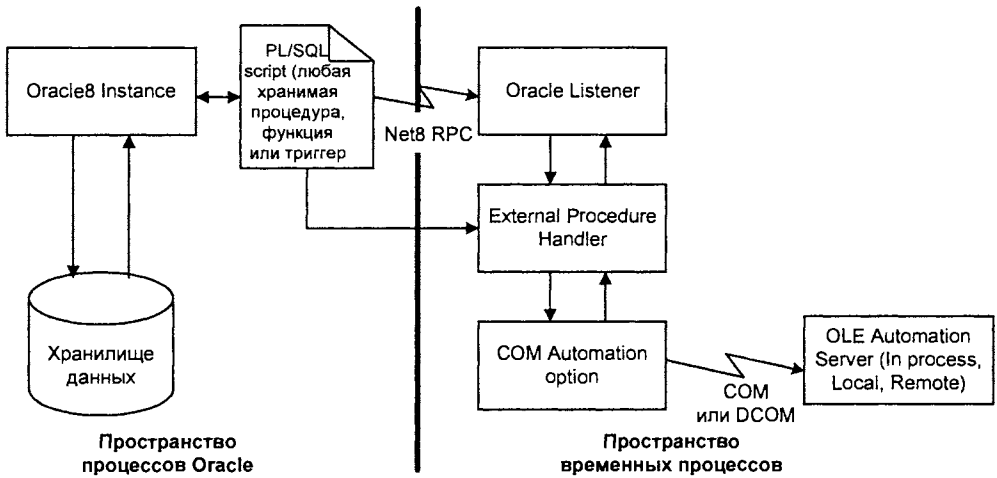


Рис. 1.19. Подключение внешних процедур

Объектно-ориентированное программирование в Oracle

Помимо удобств, предоставляемых для PL/SQL-программистов, Oracle содержит в себе средства для объектного программирования, которые могут быть вызваны из других программных сред: например, Visual Basic или Lotus Script.

С точки зрения программиста, использующего средства Lotus Domino Server, СУБД Oracle представляется как один огромный OLE-контейнер, к методам которого для извлечения данных из СУБД можно обращаться непосредственно из форм Lotus Notes, программируя в них специальные кнопки и привязывая к ним соответствующие скрипты, использующие объектную модель Oracle (рис. 1.20).

Чтобы использовать все преимущества этой модели, Lotus Notes-программист может использовать в скриптах на LotusScript объектные спецификации Oracle, подробно описанные в документации по RDBMS, например:

- в некоторой форме по нажатию кнопки в форме Lotus Notes организуется сеанс связи с Oracle и извлекаются данные из соответствующих таблиц;

- затем эти данные, естественно, могут быть помещены куда угодно, т. е. в любые определенные поля базы данных Lotus Notes (рис. 1.21).

Рис. 1.20. Уровни программного обеспечения 0040 — схема взаимодействия объектов

Текст скрипта:

```
Sub Click(Source As Button)
     Dim OraSession As Variant
     Dim OraDatabase As Variant
     Dim OraDynaset As Variant
     'Создание OraSession Object.
     Set OraSession = CreateObject("OracleInProcServer.XOraSession")


    'Создание OraDatabase-объекта при открытии соединения с Oracle.
     Set OraDatabase = OraSession.OpenDatabase("ExampleDb",
"scott/tiger", 0&)


    'Создание OraDynaset-объекта.
     Set OraDynaset = OraDatabase.CreateDynaset("select * from emp where
ename='Дунаев'", 0&)
     Messagebox (OraDynaset.RecordCount & " записей извлечено из базы.")
     Messagebox ("Первое имя в первой строке — " &
                                        OraDynaset.Fields("ENAME").Value)
```

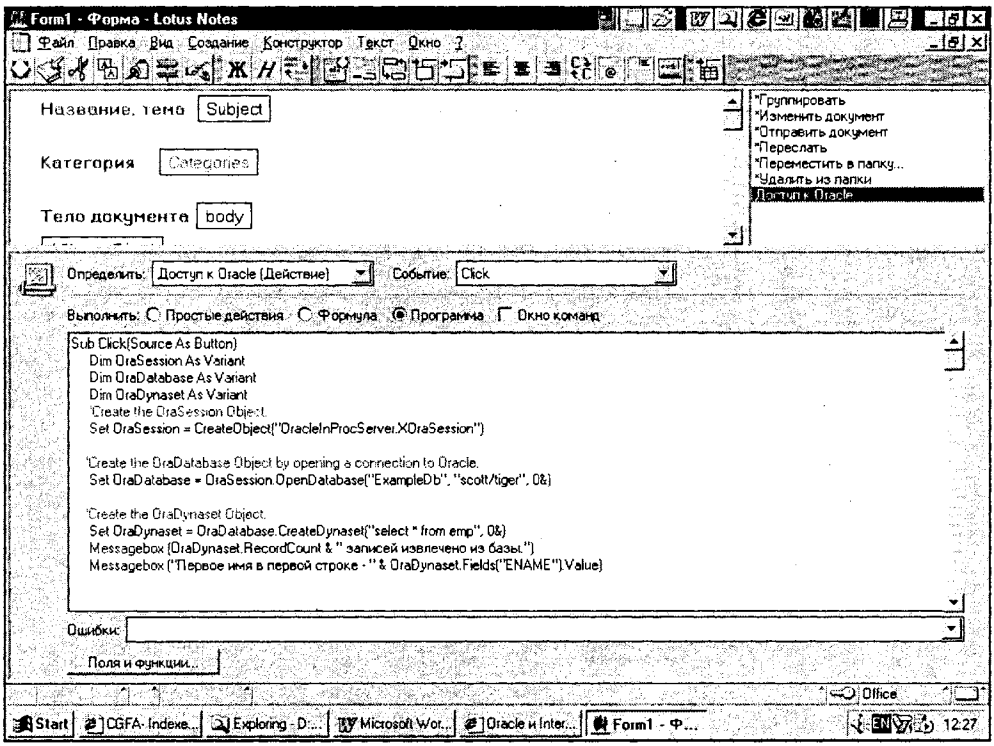


Рис. 1.21. Форма Lotus Notes, из которой по нажатию управляющей кнопки программируется событие на LotusScript, связанное с нажатием этой кнопки

ActiveX-документы и документы-контейнеры типа MS Internet Explorer

Другим мощным и удобным средством построения интерактивных Интернет-приложений является технология Microsoft, получившая название ActiveX Documents. Шаблоны этой категории в Visual Basic (ActiveX Document EXE и ActiveX Document DLL) обеспечивают создание программ, которые являются специальными типами объектов ActiveX и могут быть помещены и активизированы внутри контейнеров документов ActiveX. Вы сами можете создавать такие контейнеры, но, как правило, это совершенно не нужно — у вас уже есть один замечательный и универсальный контейнер — MS Internet Explorer, начиная с версии 3.0 и выше. Помните, как было удивительно, когда мы наблюдали, как MS IE 3.0 встраивал в себя документы типа MS Word (.doc) или Acrobat (.pdf). С развитием ActiveX-технологий и RAD-систем программирования, способность встраивания в браузер получили любые программы, если они построены по правилам документов ActiveX (ActiveX Documents).

Обогащение OLE-технологий на новом витке спирали развития Интернета

Технология документов OLE и раньше позволяла внедрять объекты внутри приложения-контейнера. При этом, исходный держатель объектов называется OLE-сервером. Так запросто, можно было внедрить объект Word в другое приложение типа MS Excel и управлять им, однако функции ActiveX-документов гораздо шире, потому что они используют дополнительные интерфейсы, позволяющие применять все инструменты сервера в приложении клиента. Возьмем, к примеру, стандартный документ Word, в который внедрен документ Excel (рис. 1.22).

Редактировать из Word внедренный объект можно только при помощи двойного щелчка мышью по внедренному объекту и переходу в программу MS Excel. В то же время, если открыть этот объект из другого контейнера документов, который поддерживает технологию ActiveX Documents — имеется в виду MS Internet Explorer 5.0, то вы увидите, что объект MS Excel появился в поле браузера со всеми элементами управления и полным набором функций, точно так же, как если бы он был вызван из родной оболочки Excel (рис. 1.23).

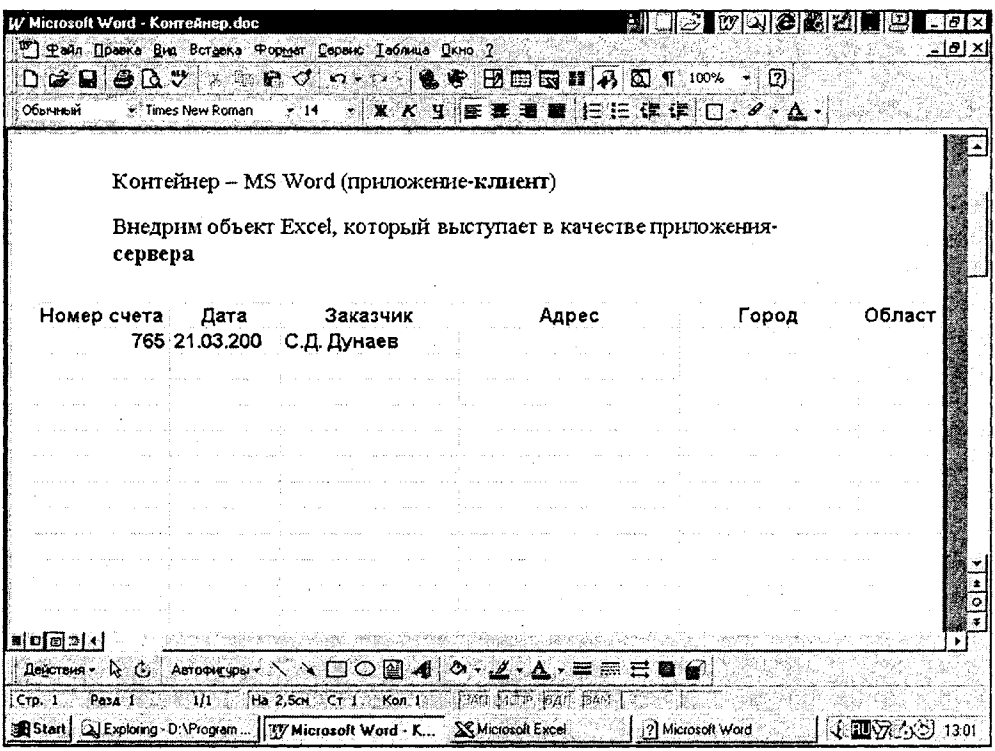


Рис. 1.22. Стандартный документ Word с внедренным документом Excel

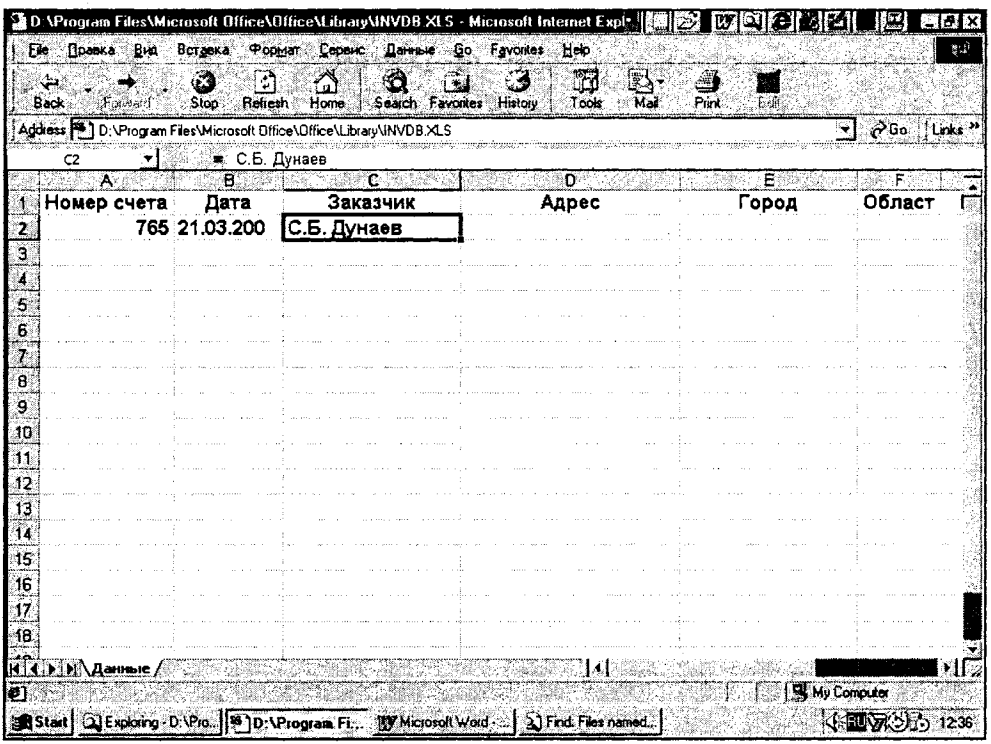


Рис. 1.23. Документ Excel в поле браузера

Но и это еще не все! Теперь, пользуясь этой замечательной технологией вы можете без труда переводить в Интернет/интранет любые готовые приложения, написанные, к примеру на Visual Basic. Не верится? А вы попробуйте. Возьмем известный пример FirstApp, входящий в любой комплект Visual Basic, и заставим работать его из браузера, не меняя в нем ни одной строчки кода. Как это сделать? Опять, как и всегда у Microsoft, для облегчения жизни пользователей в VB 6 появилось много замечательных мастеров (Wizards). Один из них — VB 6 ActiveX Doc Migration Wizard. Если выбрать какой-нибудь стандартный EXE-проект, разработанный для выполнения на персональном компьютере и даже использующий таблицы СУБД, то его можно с помощью этого мастера легко превратить в Интернет-приложение. Мастер-волшебник задаст всего несколько вопросов, на которые легко ответить, и преобразует шаблон приложения так, что он из обычного приложения превратится в приложение ActiveX EXE, способное работать в Интернете. При этом мастер добавит объект `UserDocument` в преобразованный проект подготовки приложения. С объектом `UserDocument` будут связаны два файла:

- **файл с расширением .dob** — текстовый файл, содержащий исходный текст и значения свойств объекта `UserDocument`, а также элементов управления,

имеющихся в форме (например, элемент управления таблицей СУБД, включающий описание ее местоположения и т. д.);

- **файл с расширением .dox** — в этом файле хранятся все графические элементы и ресурсы, например рисунки, требующиеся для оформления элементов управления.

Кроме этих двух файлов в папке проекта при его преобразовании к виду ActiveX (.dll или .exe — в зависимости от предназначения приложения — внешнее или внутреннее) будет создан еще один очень важный файл с расширением .vbd (Visual Basic Document). Как правило, он формируется с именем главной формы и служит в основном только затем, чтобы именно его указывать в качестве имени приложения, вызываемого на выполнение из браузера. Для проекта FitstApp — это будет файл с именем docform1.vbd (рис. 1.24).

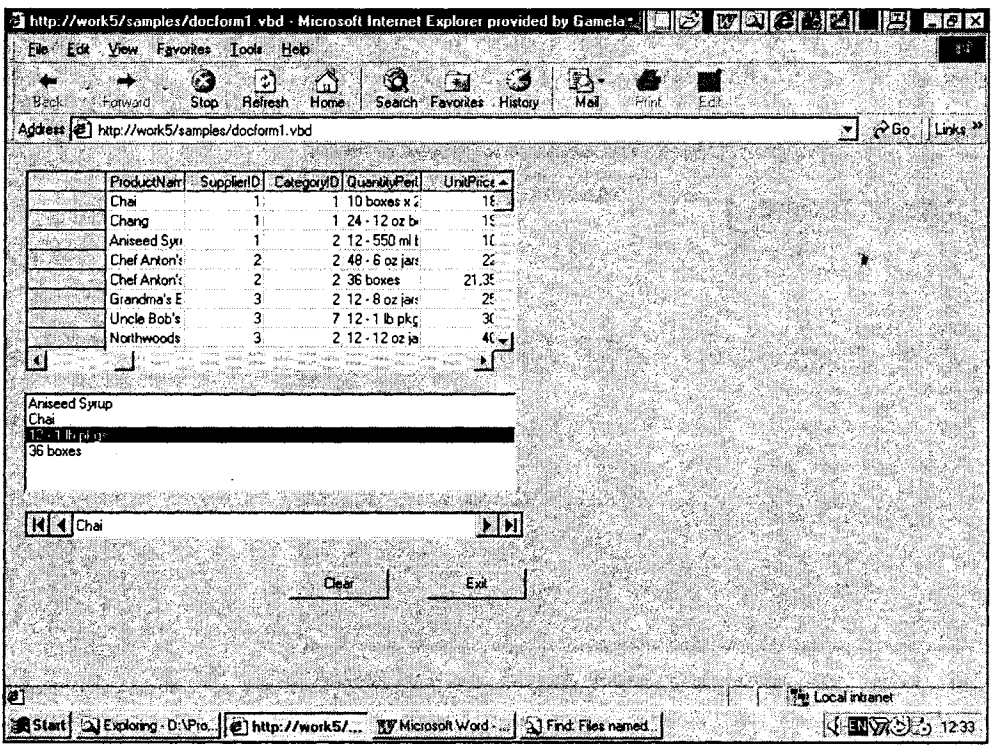


Рис. 1.24. Файл docform1.vbd в окне браузера

Что при этом происходит? Браузер загружает ActiveX-приложение в свое пространство, выделяя ему заданный прямоугольник в окне, и предоставляет фактический карт-бланш, в том смысле, что приложение получает полный контроль над компьютером, так, как если бы оно выполнялось не в

браузере, а в реальности было запущено на том компьютере, куда приложение было загружено. В этом, конечно же, таится очень большая опасность. Следует помнить, что браузер — это вовсе не графическая оболочка для исполнения команд и программ, что-то вроде графического Shell в Unix. Нет, в понимании Microsoft нынешний браузер является неотъемлемой частью операционной системы. Ведь в точности таким же способом, каким вы только что подготовили полнофункциональное приложение для исполнения его через браузер, вы можете загрузить через Интернет какое-нибудь другое приложение, которое, с точки зрения архитектуры и внутренней структуры, ничем не будет отличаться от вашего, но может скрывать в себе потенциальную угрозу для компьютера. Браузер имеет несколько уровней защиты и никогда не будет загружать по умолчанию неподписанный файл. Посмотрим, каким способом можно подготавливать ваши приложения для развертывания и эксплуатации их в сети Интернет/интранет.

В Visual Basic для этих целей существует специальный мастер под названием Package and Deployment Wizard. Советую сразу выбрать в режиме **Load on Startup** всех необходимых мастеров (рис. 1.25).

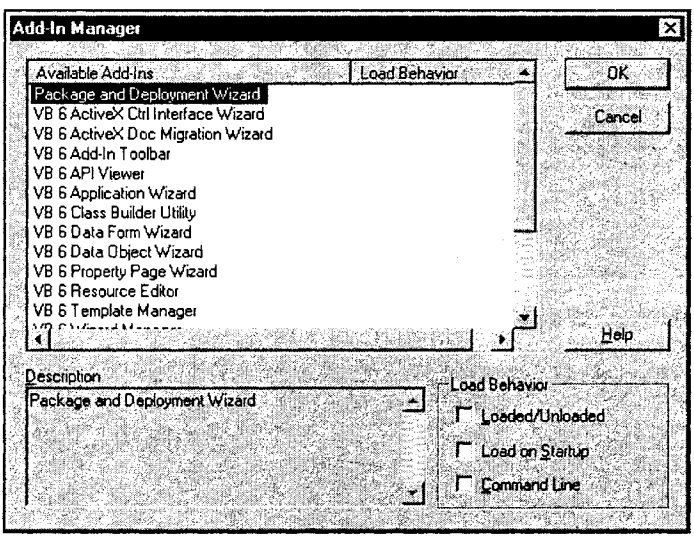


Рис. 1.25. Окно для выбора необходимого мастера (Wizard)

В результате работы этого мастера создается инсталляционная процедура, которая копирует все необходимые для работы приложения библиотеки и драйвера (включая даже ODBC) и упаковывает все это в файл с расширением .cab, который можно загружать по сети из браузера. В момент загрузки, если на компьютере нет требуемых библиотек и драйверов, они создадутся и будут проведены все необходимые процедуры регистрации. Пользователю останется только одно — нажимать на кнопки и отвечать на вопросы — до-

веряет или нет он тому сайту, с которого производится загрузка приложения. На первый взгляд Microsoft продумала все идеально. Но это только на первый взгляд. На самом деле даже приложение FirstApp, которое мы за считанные минуты превратили в полноценное Интернет-приложение, требует весьма значительных ресурсов по времени загрузки, особенно в тех случаях, если оно грузится на компьютере, где даже не установлен Visual Basic и, следовательно, отсутствуют многие важные файлы OCX, DLL и пр. Именно из них (элементов управления) и строится большая часть приложений. Конечно, в этом нет ничего страшного — CAB-файл содержит в себе все необходимое и при его разворачивании на компьютер клиента грузится только то, что у него отсутствует. Тем не менее мастер Package and Deployment для приложения FirsApp, в том случае, если бы его захотели эксплуатировать по Интернету, создал бы файл объемом в 7 990 Кбайт. Кроме этого он создал бы еще кучу полезных файлов, например, файл FIRSTAPP.HTM для запуска приложения из браузера:

```
<HTML>
<HEAD>
<TITLE>FIRSTAPP.CAB</TITLE>
</HEAD>
<BODY>

<a href=docForm1.VBD>docForm1.VBD</a>
<!--*********** Comment Begin **********
    Internet Explorer Version 3.x HTML
    ==================================
    The following HTML code has been commented
    out and provided for ActiveX User Documents
    download support in IE 3.x only.  This
    HTML script may not work properly in later
    versions of Internet Explorer.

    Additional information about downloading
    ActiveX User Documents in IE 3.x can be
    found in Microsoft's online support on the
    internet at http://support.microsoft.com.
    *********** Comment End   ********** -->

<!--*********** Comment Begin **********
<HTML>
<OBJECT ID="docForm1"
CLASSID="CLSID:"
CODEBASE="FIRSTAPP.CAB#version=6,0,0,8132">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Sub Window_OnLoad
```

```
    Document.Open
    Document.Write "<FRAMESET>"
    Document.Write "<FRAME SRC=""docForm1.VBD"">"
    Document.Write "</FRAMESET>"
    Document.Close
End Sub
</SCRIPT>
</HTML>
    *********** Comment End   ********** -->

</BODY>
</HTML>
```

Стоит какому-нибудь пользователю запустить из своего браузера с моей Web-странички этот файл, то он, после небольшой паузы, увидит на своем экране то же, что вижу я в результате работы приложения, а именно распечатку и таблицу с данными (см. рис. 1.24). Однако и тут кроются подводные камни. А что будет, если требуемая таблица с данными отсутствует на вашем компьютере? Браузер в этом случае выдаст предупреждающее сообщение и нарисует пустые клетки. Но ведь я таблицу не экспортирую. Конечно, можно экспортировать и ее или, по крайней мере, ее реплику, однако более предпочтительным представляется вариант, когда работать с таблицей надо непосредственно на сервере, никуда ее не перетаскивая. В таком случае, зачем вообще к себе что-то грузить? Не лучше ли, чтобы и все программы исполнял сервер? Но тут мы уже вступаем в область принципиально иных подходов и технологий, которые имеются и у Microsoft, и у соперничающих с нею фирм. Об этих подходах и технологиях речь пойдет в следующих разделах книги, а пока скажем еще несколько слов об Active Desktop.

Активная платформа и ActiveX

В последнее время имеется еще одно название для обозначения всего комплекса технологий ActiveX — "Активная платформа", включающего три основные технологии — Active Desktop, Active Server и ActiveX. Active Desktop состоит из услуг, которые, по утверждению Microsoft, будут интегрированы во все версии Windows и другие главные операционные системы, включая MacOS и Unix. Эта интеграция будет происходить через MS Internet Explorer, играющего роль троянского коня. Конечно, интеграция на этом уровне не такая же, как интеграция с операционной системой непосредственно. Active Server — это набор API-интерфейсов, которые будут обеспечены MS Internet Information Server 3.0/4.0, работающего под Windows NT.

ActiveX расширяет существующую модель связи и внедрения объектов Microsoft (OLE) и компонентную объектную модель (COM) по отношению к

локальным и глобальным сетям. Подобно JavaBeans, ActiveX служит каркасом для объединения и сборки компонентов; в этом смысле, данные две технологии являются прямыми конкурентами.

Клиентская технология ActiveX (Active Desktop)

ActiveX реализуется на машине-клиенте с помощью библиотек, поставляемых вместе с Internet Explorer 4.0/5.0. В дальнейшем эти библиотеки будут дополняться и переписываться, в частности, наиболее значимые обновления этих библиотек на клиенте следует ожидать после выхода NetShow, продукта, предназначенного для оптимальной передачи по сети данных мультимедиа.

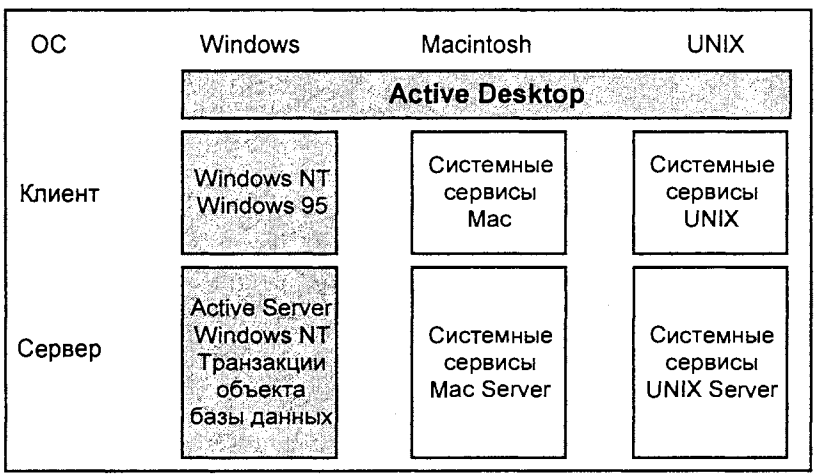


Рис. 1.26. Структурная схема активной платформы для разных машин и операционных систем

Программные компоненты ActiveX могут быть установлены автоматически на компьютер пользователя по сети с удаленного сервера, причем будет загружен код, подходящий для конкретной платформы клиента, будь то Macintosh, Windows или Unix (рис. 1.26). Разработчик Web-страниц может либо сам запрограммировать элементы ActiveX, используя популярные языки программирования Visual C++, Visual Basic или Java, либо использовать существующие. Примеры готовых программных элементов можно найти по адресу **www.microsoft.com/activeplatform/default.asp**.

Используя языки сценариев ActiveX, программисты могут обеспечить взаимодействие различных элементов ActiveX, Java, других программ на клиентском компьютере и различных частей самого Internet Explorer. Например, программный элемент синхронизации может обновлять страницу Web через

определенные промежутки времени. Можно также периодически запускать программный элемент, привлекающий внимание пользователя. Имеются реализации Visual Basic Scripting Edition, являющегося подмножеством Visual Basic и JScript. Кроме того, разработчик может написать интерпретатор собственного языка сценариев и добавить его в систему.

С ActiveX Documents знаком каждый, кто работал с составными документами. С помощью Internet Explorer можно работать, например, с таблицами Microsoft Excel и файлами других офисных приложений. Это делает программу просмотра универсальным средством, способным не только отображать файлы в формате HTML и осуществлять переходы по ссылкам, но и поддерживать работы с документами любых приложений и даже запуск программ.

Если сравнивать все имеющиеся компоненты из арсенала клиентской части ActiveX, то, все-таки, нельзя не отметить, что программисты на Java находятся в некотором преимуществе, поскольку могут создавать на своем языке ActiveX-элементы точно так же, как это можно делать из Microsoft Visual Basic (VB) и Visual C++ (VC++).

Серверная технология ActiveX (Active Server)

Серверная часть технологии ActiveX реализована с помощью Microsoft Internet Information Server 3.0/4.0/5.0, а также с использованием MTS и MSMQ. ActiveX позволяет писать программы, выполняющиеся на сервере, на языках сценариев (сейчас это VBScript). Если раньше разработчикам приходилось применять такие средства, как Microsoft Visual C++ для написания динамически загружаемых библиотек, использующих специальные вызовы Internet Server API, то теперь возможно написание приложений на языке сценариев. Это существенно упрощает разработку, сокращает время написания программы и минимизирует затраты. Программы, основанные на технологиях Active Server, на порядок производительнее программ, основанных на Common Gateway Interface (CGI). Это достигается оптимизацией процессов ActiveX на сервере, учитывающих архитектуру Windows NT.

С помощью языков сценариев на сервере можно осуществлять доступ к системам управления базами данных, поддерживающим стандарт ODBC, и использовать механизм транзакций.

Поскольку подход к применению технологий ActiveX на сервере стандартизован, программисты могут не только разрабатывать приложения, способные выполняться на серверах, но и реализовывать свои схемы взаимосвязи серверных приложений и сервисов, создавать собственные интерпретаторы серверных языков сценариев.

Дополнительные средства от Microsoft для создания распределенных приложений

Еще до выпуска Windows 2000 в состав Windows NT 4.0 Enterprise Edition вошли Microsoft Transaction Server 1.1 (MTS) и Microsoft Message Queue Server 1.0 (MSMQ). Microsoft Transaction Server сочетает в себе функции монитора транзакций и брокера объектных запросов и предназначен для управления компонентами приложения, вынесенными в промежуточное между клиентом и сервером звено.

Приложение, построенное из компонентов, имеет массу преимуществ по сравнению с традиционным программированием. Главное достоинство состоит в возможности распределения разных компонентов по разным компьютерам. Тем самым достигается балансировка нагрузки, и вычислительные ресурсы сети используются максимально эффективно. Затраты на создание компонентных приложений существенно ниже, т. к. компоненты, реализующие основные прикладные функции, могут повторно включаться в различные приложения. Благодаря тому, что разные компоненты могут создаваться на разных языках программирования, отпадает необходимость переводить всех разработчиков в проекте на единый инструментарий — каждый вправе использовать то средство, в котором он наиболее уверенно себя ощущает (лишь бы оно позволяло создавать компоненты) и, следовательно, может добиться наибольшей эффективности. Microsoft Transaction Server позволяет быстро разрабатывать такие приложения, обеспечивая их согласованную работу и транзакционную целостность.

Таким образом, выгоды от применения MTS одинаково очевидны как для разработчиков, так и для администраторов. С точки зрения разработчиков многоуровневых приложений применение MTS обусловлено необходимостью поддержки транзакционной целостности на уровне приложения. С точки зрения администратора MTS хорош тем, что берет на себя автоматическое управление системными ресурсами: процессами, потоками, памятью, соединениями с базами данных, экземплярами компонентов и т. д. Известно, что дефицит ресурсов крайне негативно сказывается на масштабируемости, т. к. при возрастании числа клиентов они вынуждены либо подолгу простаивать в ожидании, либо попросту отваливаться. Применение MTS гарантирует высокую масштабируемость приложений в многопользовательской среде притом, что сами компоненты могут разрабатываться как однопользовательские.

Возможности Microsoft Transaction Server

MS Transaction Server поддерживает структуру многоуровневых приложений и компоненты. Клиенты первого уровня обеспечивают пользовательский интерфейс и некоторое количество логики приложения. Серверы приложений среднего звена берут на себя большую часть логики приложения. Они

работают под управлением Transaction Server. Базы данных конечного звена занимаются хранением информации для приложения.

Transaction Server предполагает, что приложения и подход к их построению являются компонентными. Считается, что компоненты дают программистам наилучший способ создания распределенных приложений. Их объектно-ориентированные свойства, такие как модульность, инкапсуляция и наследование, позволяют легко расширять приложения и облегчают поддержку. Следовательно, при разработке они легко используются повторно, а также упрощают разбиение приложения и его распределение.

COM и ActiveX

Transaction Server поддерживает Microsoft Common Object Model (COM) и технологию ActiveX. Модель COM обеспечивает "соединяемость" и определяет интерфейсы и протоколы внутри и между компонентами приложения. Разработчики проектируют взаимодействие составных частей приложения на основе ActiveX-модулей. Таким образом, упрощенно можно представлять Transaction Server как законченную двухуровневую серверную среду для ActiveX.

Когда пользователь или программа ссылается на интерфейс компонента ActiveX посылая сообщение или событие, COM определяет компонент и перенаправляет ему данное сообщение или событие. Компонент, на который происходит ссылка, может быть локальным или находиться на удаленном компьютере. *Локальная соединяемость* обеспечивается механизмами межпроцессорного взаимодействия самой операционной системы. *Удаленная соединяемость* обеспечивается протоколом TCP/IP.

Разработчикам остается только вставить в своем коде ссылку на соответствующий объект ActiveX. Все остальное сделают COM и DCOM (распределенное сетевое расширение COM). Описанные ниже сервисы службы каталогов предоставляют COM и DCOM всю необходимую информацию для разрешения ActiveX-ссылок.

Transaction Server задействует механизмы безопасности Windows NT. Windows NT обеспечивает вход в систему через пользовательский ID и пароль, а также управление доступом к ресурсам с помощью администрирования пользователей и групп пользователей. Transaction Server расширяет эти возможности за счет управления доступом к индивидуальным компонентам приложения. Контроль может выполняться как с помощью административных функций Windows NT по отношению к пользователям и группам, так и программным путем внутри компонентов приложения.

Службы каталогов обеспечиваются в Transaction Server и COM через реестр Windows (Registry), который содержит информацию о пользователях, группах, приложениях, именах и интерфейсах ActiveX. Он предоставляет все не-

обходимое для поддержки многоуровневых приложений в Microsoft Windows. Создание и поддержка реестра является административной функцией, разработчики избавлены от необходимости следить за этим.

Разработчики своих приложениях просто пишут код на SQL для получения информации.

Transaction Server имеет монитор транзакций, который контролирует доступ транзакций к менеджерам ресурсов. Транзакции могут обращаться к единственному менеджеру ресурсов, или через поддержку протокола Microsoft Distributed Transaction Coordinator (DTC) транзакции могут согласовывать и синхронизировать свой доступ к нескольким менеджерам ресурсов. В начальной версии Transaction Server в роли менеджеров ресурсов могут выступать Microsoft SQL Server и базы данных, поддерживающие интерфейс ODBC. Новые версии будут поддерживать протоколы XA, SNA LU6.2 и TIP (Transaction Internet Protocol) и, следовательно, будут расширены понятия участвующих в транзакциях менеджеров ресурсов и среды обработки.

Программистские усилия по поддержке транзакций сводятся к минимуму. Пишущим код не нужно вставлять в программу операторы `Begin transaction`, `End transaction` или `Abort transaction`. Транзакции определяются путем задания свойства компонента приложения. Если компонент помечен как "transactional", Transaction Server будет управлять транзакциями во время всего периода его выполнения. Все компоненты, на которые она ссылается, автоматически принимают участие в транзакции. Если компоненты помечены как транзакционные, то все их выполнение будет заключено внутри единой транзакции. Shared Property Manager (менеджер разделенных свойств) в составе Transaction Server предоставляет механизм разделения доступа к глобальной информации между серверными процессами. Разработчики могут его использовать для широкого круга вопросов, включая мониторинг и управление исполнением, чередование процессов и т. д. Shared Property Manager выполняет все необходимые действия по разделению информации между приложениями. Разработчики могут спокойно концентрировать усилия на программной логике. Поскольку действия выполняются централизованно, процесс отладки и сопровождения распределенных приложений проходит более гладко, нежели в тех случаях, когда созданием и управлением глобальными данными занимаются отдельные разработчики.

Почти два года тому назад, 26 июня 1998 года было объявлено, что MTS 1.1 становится атрибутом операционной системы Windows NT 4.0. Тогда же были внесены соответствующие изменения в текст лицензионного соглашения для пользователей Windows NT 4.0 и BackOffice 2.5. Лицензированные пользователи получили возможность "скачать" MTS 1.1 с Web-сайта Microsoft. Версия 1.1, по сути, представляет собой базовую версию 1.0, объединенную с кумулятивным сервисным пакетом 2.0a, который содержал в основном интерфейсные исправления типа "при нажатии клавиши <F1> в MTS

Explorer не вызывалась справочная система Help". В MTS добавились функции интеграции с мониторами транзакций на мэйнфреймах IBM (проект Cedar) и улучшенные возможности работы с кластерами MSCS.

MTS обеспечивает плотную целостность распределенных транзакций по протоколу двухфазной фиксации. Тем не менее существуют ситуации, когда более выгодно работать асинхронно. К ним относятся приложения, в которых инициатор транзакции должен продолжать работу, не дожидаясь результатов обработки запроса. Приложения, обменивающиеся сообщениями, могут не быть постоянно в режиме on-line. Наконец, требования асинхронной коммуникации могут диктоваться плохими каналами связи, мобильными пользователями и т. д. Во всех этих случаях для обеспечения асинхронного взаимодействия необходимо иметь некий механизм промежуточного хранения и диспетчеризации пересылки (store and forward).

К преимуществам подобного подхода следует отнести гарантированную доставку, возможность маршрутизации сообщений и назначения им приоритетов и известную независимость от качества каналов связи. Такой механизм разрабатывался в Microsoft под кодовым названием Falcon и вошел в Windows NT 4.0 Enterprise Edition как сервер очередей сообщений — Microsoft Message Queue Server (MSMQ 1.0). Под сообщением MSMQ понимается запрос произвольной природы (или ответ на запрос), самодостаточный в плане координат отправителя/получателя и содержания. Сообщения, запрашивающие участие менеджера ресурсов в транзакции, попадают в очереди MSMQ. Очереди сообщений схожи с почтовыми ящиками сервера электронной почты. Менеджер очередей MSMQ также работает подобно серверу электронной почты, решая, кому, когда и как доставить сообщение. Топология MSMQ напоминает структуру Microsoft Exchange Server: единая распределенная база данных на Primary Enterprise Controller (PEC) предоставляет в общее пользование информацию о доступе к сообщениям для всех компьютеров, на которых запущен MSMQ.

Под узлами (sites) MSMQ понимаются группы компьютеров, соединенных надежными линиями связи (т. е., как правило, объединенных в локальную сеть). Внутри каждого узла выделяются Primary Site Controller (PSC) и Backup Site Controller (BSC). Таким образом, Windows NT PEC соответствует Primary Domain Controller, а PSC и BSC — Backup Domain Controller. Узлы могут быть связаны друг с другом по линиям ATM, Frame Relay, ISDN и т. д. Каждому ненадежному соединению назначается условная стоимость, например, исходя из времени задержки доставки сообщения. Внутри узлов стоимость полагается равной нулю. MSMQ Enterprise определяет минимальный по стоимости маршрут доставки, а в случае его выхода из строя находит обходные пути. При недоступности адресата служба управления очередями сообщений MSMQ предпринимает (если это оговорено в свойствах) повторные попытки передать сообщение, гарантирует, что оно будет доставлено

только один раз, и может прислать подтверждение. Еще одно преимущество MSMQ состоит в управлении приоритетами сообщений, по сравнению со стандартным методом FIFO, который обычно применяется в серверах баз данных.

Несмотря на достаточно близкие аналогии с электронной почтой, MSMQ имеет несколько существенных отличий. Он разрабатывался специально для взаимодействий "приложение-приложение", содержание сообщения имеет значение только для его отправителя и получателя, существует возможность настройки гарантии доставки (экспресс, гарантированная и транзакционная), наконец, MSMQ является более простым, компактным, быстрым (и, как следствие, более дешевым), чем традиционный почтовый сервер. Технология очередей сообщений может служить шлюзом к серверам баз данных, работающим на других операционных системах, обеспечивая многоплатформные распределенные транзакции. Поддержка MSMQ API на платформах IBM MVS и CICS, Sun Solaris, HP-UNIX, AIX UNIX, OS/2, VMS и AS/400 обеспечивается продуктами Level 8 Systems (см. **www.level8.com**). Возможно также отображение прикладного интерфейса сервера очередей IBM MQ Series в вызовы MSMQ API.

Программный интерфейс MSMQ достаточно прост: отправитель имеет возможность открыть очередь, установить режим доставки и продолжительность попыток, записать сообщения в очередь, задать их свойства, назначить очередь для приема ответа. Аналогично получатель открывает очередь, читает сообщения и т. д. Одну очередь могут читать и модифицировать при наличии соответствующих прав несколько приложений, при этом можно читать и изменять как свои, так и удаленные очереди. MSMQ имеет дополнительные интерфейсы в Microsoft Exchange Server, MAPI и RPC. MTS рассматривает MSMQ как менеджер ресурсов, что дает возможность включать операции над сообщениями серверов очередей в контекст единой распределенной транзакции.

MSMQ API поддерживается всеми тремя конфигурациями сервера очередей: MSMQ Server, MSMQ Workstation и MSMQ Client. Наиболее полной, естественно, является серверная конфигурация, куда, помимо API, входят службы маршрутизации, каталогов, менеджер очередей и среда MSMQ Explorer. MSMQ Client, напротив, является наименее мощной конфигурацией: в него не входят локальное хранилище и менеджер очередей, он пользуется этими службами у MSMQ Server, который должен быть постоянно для него доступен. Таким образом, MSMQ Client позиционируется как сетевой клиент. Рабочая станция MSMQ (MSMQ Workstation) представляет собой промежуточный вариант: у нее отсутствует поддержка маршрутизации, зато она может обслуживать локальные очереди сообщений и работать при отсутствии связи с сервером. MSMQ Workstation позиционируется как мобильный клиент.

Возможности Message Queue Server

Схема обобщенной архитектуры Microsoft Message Queue Server приведена на рис. 1.27.

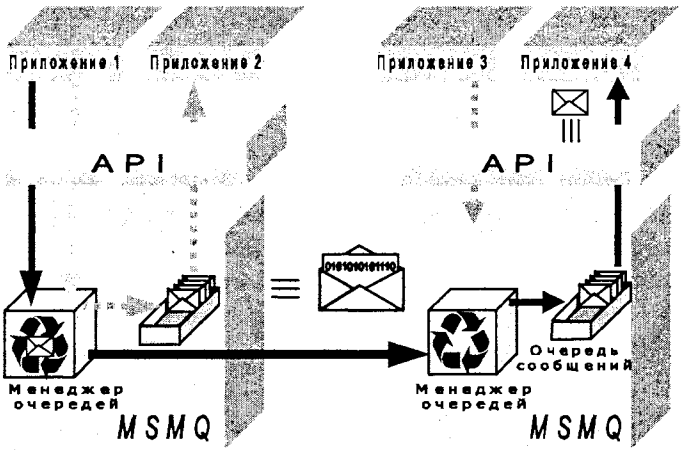


Рис. 1.27. Обобщенная архитектура сервера сообщений

Microsoft Message Queue Server 1.0 имеет перечисленные ниже возможности.

Передача сообщений при отсутствии прямых соединений (Connectionless messaging)

Наличие очередей для сохранения и передачи сообщений избавляет приложения от необходимости обрабатывать нештатные ситуации поведения сети. Сервер очередей использует модель соединения вне сеанса связи (sessionless model), поэтому не требуется поддержка адресатом и отправителем одних и тех же сетевых протоколов. Сервер очередей поддерживает IP и IPX. Кроме того, фирма Level 8 Systems предполагает обеспечить поддержку SNA для обращений к компьютерам IBM (MVS и CICS).

Установка приоритетов передачи (Network traffic prioritization)

Возможность расстановки приоритетов позволяет в первую очередь отправлять более срочные сообщения, что гарантирует хорошее время отклика для критичных приложений за счет менее важных.

Динамические очереди (Dynamic queues)

Информация об очередях хранится в базе данных, которая может изменяться динамически и реплицироваться. Подобный подход позволяет администраторам менять протокол и положение (помимо других параметров очереди), не оказывая влияния на приложения передачи/приема сообщений. Ад-

министратор может вносить указанные изменения с помощью MSMQ Explorer, где бы он не находился.

Маршрутизация (Routing)

Сервер очередей поддерживает динамическую маршрутизацию и доставляет сообщения, используя путь с наименьшей ценой доставки. Когда работа сети на этом маршруте нарушается, сервер автоматически выбирает следующий по стоимости маршрут. Назначение цены участков передачи сообщений производится администратором при помощи административной консоли. Администраторы также могут назначить серверу очередей роль коммуникационного концентратора для обработки всего графика между двумя узлами.

Гарантированная доставка (Guaranteed delivery)

Сообщения могут сохраняться в дисковой очереди для обеспечения гарантированной доставки (journaling) или уведомления о невозможности доставки (dead letter). Для очень срочных сообщений сервер очередей предоставляет возможность сохранять сообщения в памяти до того момента, когда они могут быть доставлены адресату. В последнем случае остановка сервера вызовет потерю сообщения.

Защищенность (Security)

Сервер очередей использует списки прав доступа (ACL) Windows NT для определения прав доступа к очереди, а также другие системы защиты, основанные на использовании открытых (public) ключей (система защиты основана на использовании Crypto API с RSA-провайдером). Для проверки целостности сообщений применяются контрольные суммы. Сервер очередей использует алгоритмы шифрования RC2 и RC4; по умолчанию выбирается алгоритм RC2. Для шифрования сообщения приложение просто присваивает ему атрибут "private", все остальное будет сделано автоматически.

Поддержка ActiveX

Как любой сервер, сервер очередей должен иметь открытую архитектуру. В нашем случае разработка приложений под сервер очередей средствами VB облегчается использованием трех OLE-компонентов:

- **Queue** (MQQueue) — открывает доступ к очереди;
- **Message** (MQMsg) — предоставляет доступ посланным и принятым сообщениям;
- **Directory Service** (MQDS) — используется для поиска существующих и создания новых очередей.

Все эти компоненты обеспечивают доступ к API для программирования всех функций сервера. Поддержка ActiveX позволяет обращаться к серверу оче-

редей из Microsoft Transaction Server, Internet Information Server, активных Web-страниц (ASP) и любых приложений, поддерживающих OLE-вызовы. Средствами C и C++ можно обращаться к API напрямую.

Поддержка транзакций

Разработчики могут включать операции сервера очередей в контекст транзакции при обращении к другим источникам данных, способным восстанавливать состояние, например, к серверам баз данных. Для сохранения целостности данных операции сервера очередей фиксируются или откатываются назад вместе с остальными ресурсами в пределах транзакции. Например, если приложение вносит модификации в базу данных и посылает сообщение другому приложению внутри одной транзакции, то может возникнуть ситуация, когда транзакция в базе данных должна быть отменена. При этом операция по передаче сообщения также будет отозвана сервером сообщений. Никакая операция по передаче сообщения не считается завершенной до тех пор, пока не зафиксирована транзакция, в которой эта операция участвует. Это значит, что адресату никогда не будет доставлено сообщение из отмененной транзакции. Аналогичные действия сервер очередей предпринимает при приеме сообщения. Если транзакция отменена, отменяются и все операции приема. Очень важно, что сами сообщения при этом не теряются, а возвращаются в очередь и остаются доступными для последующих транзакций.

Сервер очередей тесно интегрирован с Microsoft Transaction Server, и все вызовы от ActiveX-компонентов, участвующих в транзакциях, автоматически включаются в активную транзакцию MTS. Когда приложения используют транзакции для "упаковки" сообщений, сервер очередей гарантирует, что каждое сообщение будет доставлено только один раз и сообщения внутри транзакции будут доставлены в том порядке, в котором они были посланы.

Администрирование

Графический интерфейс администратора позволяет динамически управлять всеми компьютерами корпоративной сети с единой консоли.

Административная консоль использует разделяемую и реплицируемую базу данных для поддержания актуальной информации. Администратор всегда имеет возможность добавления, удаления, модификации и перемещения очередей. Отсутствие статических таблиц конфигурации и предопределенных путей улучшает масштабируемость и снижает затраты на администрирование системы.

Вся система администрирования сервера очередей концентрируется вокруг индивидуальных менеджеров очередей. Сообщения перемещаются между менеджерами очередей посредством *очередей передачи* (transmission queue). Менеджеры очередей группируются в *узлы* (sites) и связаны не "каждый с каждым", а через общее сетевое пространство предприятия. Информация

об узлах и связях является разделяемой и используется всеми компонентами, участвующими в процессе. Это служит гарантией того, что сообщения будут маршрутизироваться корректно и оптимально.

Конфигурация

Сервер очередей может быть сконфигурирован в одной из трех модификаций: сервер, рабочая станция и клиент (рис. 1.28).

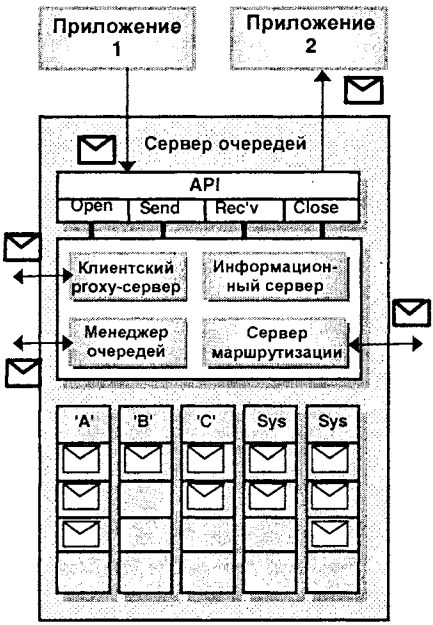


Рис. 1.28. Схема конфигурации сервера очередей

MSMQ как сервер

Сервер (рис. 1.28) поддерживает все доступные функции. Сервер способен:

- обслуживать API-вызовы от приложений, выполняющихся на той же самой машине;
- выступать как клиентский "прокси-сервер" (client proxy server) от имени клиентов сервера очередей на других машинах;
- сохранять сообщения в очередях на машине, где выполняется сервер очередей;
- выполнять функции маршрутизации сообщений;
- поддерживать соединения с другими платформами через MSMQ Connector.

Кроме того, серверная конфигурация поддерживает MSMQ Information service — каталог для информации о маршрутизации и конфигурации, и MSMQ Explorer — основную административную консоль.

MSMQ как рабочая станция

В конфигурации "рабочая станция", или MSMQ Independent Client (рис. 1.29), сервер очередей может обслуживать API-вызовы от приложений, выполняющихся на той же машине, и способен обслуживать локальные очереди сообщений, т. е. сообщения хранятся на той же машине, где он выполняется. Рабочая станция MSMQ не содержит служб маршрутизации, "прокси", или служб MSMQ Information Services. Обычно рабочая станция MSMQ выступает как клиент "полного" сервера очередей.

Рис. 1.29. MSMQ в конфигурации"рабочая станция"

Если отказала сеть или клиент вовсе не подключен к сети (например, мобильный пользователь), приложения могут продолжать отправлять и принимать сообщения из локальной очереди.

После восстановления или инициализации соединения рабочая станция MSMQ отправляет все накопленные в локальной очереди сообщения, предназначенные для другой машины. Точно так же все сообщения, предназначенные для рабочей станции MSMQ, поступают к ней в очередь с других машин. Подобная функциональность делает рабочую станцию MSMQ очень привлекательным решением, когда необходимо обеспечить хранение и периодическую передачу сообщений с обычного настольного компьютера.

Рабочая станция MSMQ не может выполнять функции маршрутизации (т. е. выступать в качестве промежуточного агента). Пересылка и прием сообщений от других рабочих станций могут выполняться только при точном указании имен очередей назначения.

Рабочая станция MSMQ может выполняться в среде Windows 95, Windows NT Workstation и Windows NT Server.

MSMQ как клиент

Конфигурация "клиент", или MSMQ Dependent Client (рис. 1.30), предоставляет все API серверу очередей (такие как Open, Close, Send, Receive) для приложений, выполняющихся на машинах, не имеющих ни сервера, ни рабочей станции для обслуживания очередей. Когда приложение выполняет API-вызов, клиент осуществляет пересылку этого вызова к клиентской "прокси"-машине, на которой исполняется серверная конфигурация сервера очередей (конфигурация "рабочая станция" не может обслуживать запросы от MSMQ-клиентов). Клиентская "прокси"-машина повторяет API-вызов к серверу очередей, который на этот раз исполняется применительно к конкретной очереди. В данной ситуации сервер отвечает за все вопросы, связанные с управлением сообщениями, хранением и маршрутизацией.

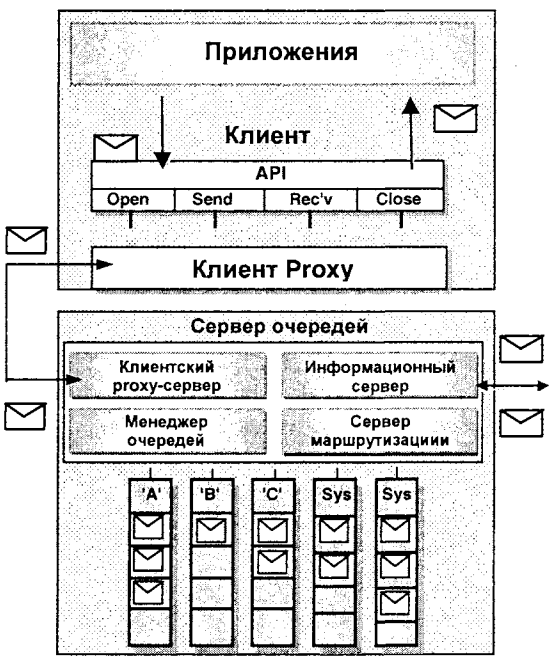


Рис. 1.30. Схема работы MSMQ-клиента

Основным преимуществом использования MSMQ-клиентов являются минимальные требования к дисковому пространству и объему памяти. Однако при отказе сети ни одно приложение не сможет выполнить вызов к API сервера.

Конфигурация "клиент" поддерживается на Windows 95, Windows NT Workstation и Windows NT Server.

В прежние времена относительно идеи выполнения одних и тех же программных модулей на различных платформах Microsoft занимала резко отрицательную позицию, но теперь она активно ратует за кроссплатформность. Почему? Microsoft поняла значение Интернета в жизни мирового сообщества и принялась активно адаптироваться к меняющимся условиям, внедряя поддержку для протоколов Интернета в свои продукты. Недавно Microsoft также высказалась в поддержку открытых стандартов. Фактически, Microsoft направила части ActiveX в Open Group (TOG), в комитет X/Open и Open Software Foundation. Многие наблюдатели интерпретируют это как желание Microsoft всяческим образом расширить применение своего Active Desktop, оставляя только за собой право совершенствования и модификации API и DCOM, которые являются первичными компонентами всего корпуса технологий ActiveX.

Это приводит к проблеме использования ActiveX на различных платформах. Хоть Microsoft и обещает сделать ActiveX кроссплатформенной, никто ни минуты не сомневается, что предпочтения самой фирмы всегда будут оставаться на стороне Windows. То есть Windows всегда будет выполнять некоторые функциональные вещи лучше, чем реализации технологии ActiveX для MacOS и Unix, которые останутся неподдерживаемыми (undersupported) и лишенными многих из замечательных возможностей?

Приложение к главе 1. Интернет-литература, бесплатные ресурсы и источники

Ниже приведены документы и стандарты, касающиеся Web-технологий и Интернета.

Консорциум W3

Наиболее представительная организация, с которой сотрудничает большинство известных фирм и все без исключения компьютерные гиганты.

http://www.w3.org

Порталы известных компьютерных фирм (IBM, Microsoft, Oracle, Lotus и их русские зеркала)

http://www.microsoft.com

http://www.microsoft.com/rus

http://msdn.microsoft.com/default.asp

http://www.lotus.com

http://www.lotus.com/russia

http://www.ibm.com/developer

http://www.oracle.com

Войдя в эти информационные порталы и двигаясь по их указателям, используя встроенные в них поисковые средства и навигаторы, можно найти практически все: любые ответы на любые вопросы, концептуальные документы, описания архитектур, тысячи всевозможных примеров, так называемые красные и белые книги, представляющие собой развернутые описания тех или иных технологий, связанных с Windows и Microsoft и не только. Microsoft предоставляет в распоряжение разработчиков MSDN Online — наиболее полный и постоянно обновляемый контент в Интернете с подробнейшим описанием всех продуктов, выпускаемых Microsoft и их пользовательской и системной документацией. Известно, что с недавнего времени Microsoft готовит свою документацию исключительно в Интернет-форматах, поэтому любой документ по любому продукту может быть получен с портала Microsoft через браузер.

Журнал "Dr. Dobbs"

Наиболее компетентный и авторитетный журнал среди разработчиков программного обеспечения. Включает статьи и исходные коды, подготовленные

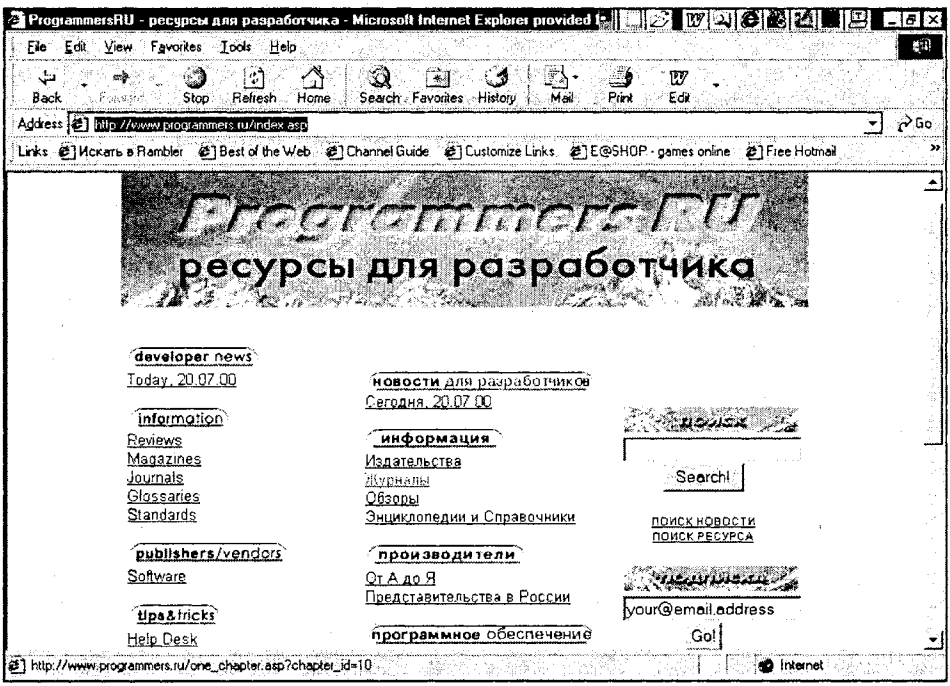
звездами программирования и авторами самых популярных программных продуктов.

http://www.ddj.com

Общероссийский сайт для разработчиков программного обеспечения

http://www.programmers.ru/index.asp

Включает достаточно много статей, исходных текстов, бесплатного ПО и т. д., преимущественно ориентированных на MS-пользователей. Постоянно обновляется.



Библиотека программиста

http://bib.ru

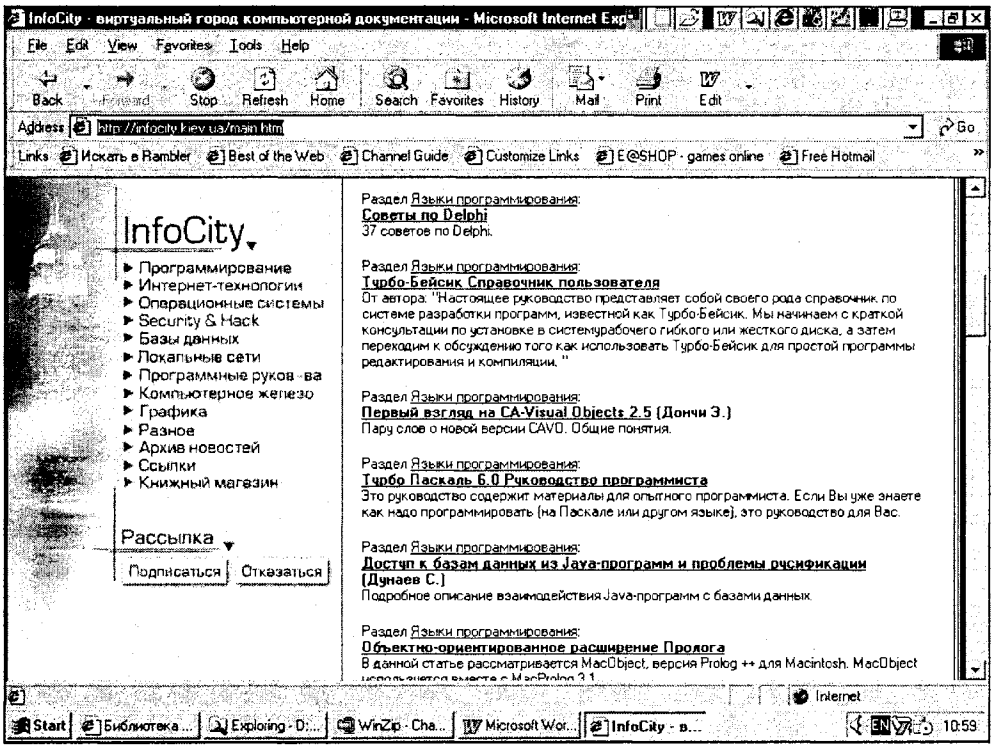
Включает множество форумов и различных материалов по C++, Delphi, Pascal, VBasic. Поощряются и премируются авторы, приславшие наиболее интересные и ходовые исходные тексты.

Виртуальный город компьютерной документации

На этом киевском сайте размещен огромный набор компьютерной документации по всем ее разделам, но разработчики, ориентированные на Microsoft,

найдут здесь очень много полезных для себя вещей, необходимых им для работы.

http://infocity.kiev.ua/main.html



ZDNet.com и его российский аналог ZDNet-Россия

http://www.zdnet.ru

http://www.zdnet.com

Крупнейшие компьютерные сайты, содержащие самую разнообразную информацию о программировании и программных средствах, включая исходные тексты, статьи, учебники и т. д. Особенной популярностью на этих сайтах пользуются разделы Developer и Download.

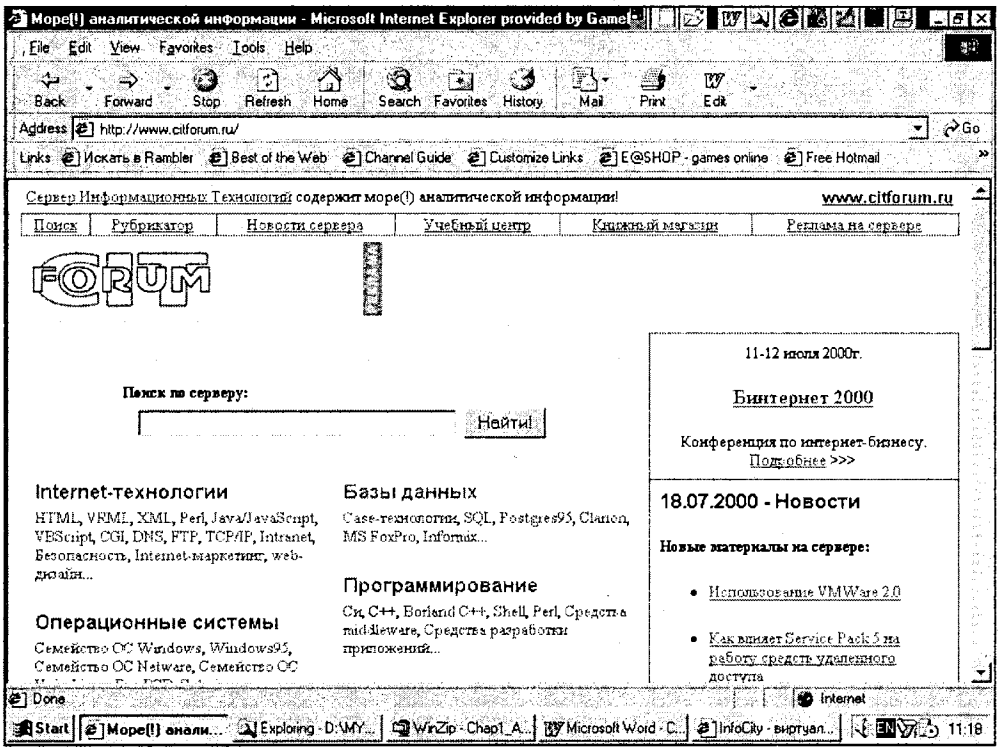
Сервер информационных технологий

http://www.citforum.ru/

Наиболее крупная и авторитетная в России компьютерная библиотека, авторы которой под эгидой МГУ тщательно проверяют сведения, и в библиотеку помещаются только проверенные временем и практикой высококачествен-

ные материалы по самым различным областям современных компьютерных технологий и программирования.

Имеет, кстати, самый высокий рейтинг, согласно TOP-List, среди компьютерных изданий в Интернете.



Профессиональный сайт, посвященный Visual Basic

http://vbwire.com/advanced

Красиво и ненавязчиво оформленный, очень умный и полезный ресурс, в котором можно найти все, что нужно, чтобы сделать на VB то, что требуется. А чего нельзя сделать на Visual Basic?

Мировой сайт для профессиональных программных разработчиков

http://developer.earthweb.com/

Пожалуй, это наиболее неисчерпаемый сайт, который содержит столько рубрик ресурсов, что в них можно захлебнуться и, вдобавок, по каждой руб-

рике — кучу полезных вещей: документов, "исходников", дистрибутивов, документации и прочего, и прочего.

Мир открытых систем

http://www.osp.ru

Наиболее известный сайт, объединяющий on-line все журналы off-line, которые имеют отношение к компьютерным технологиям, проектированию систем и программированию, в том числе и такие культовые издания, как "СУБД" и "Открытые Системы". В настоящее время на этом сайте размещаются оффлайновые версии популярных журналов:

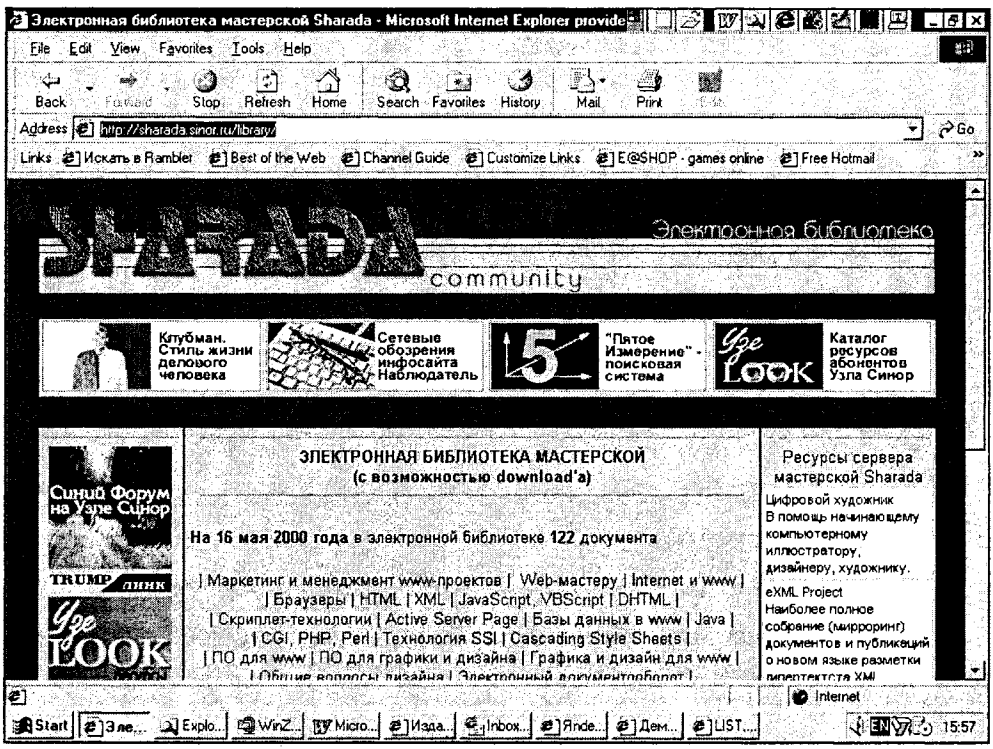
- "ОС"
- "Publish"
- "Windows-2000"
- "Computerworld"

- "Мир ПК"
- "SQL Server Online"
- "Сети"
- "LAN"

Электронная библиотека мастерской Sharada

http://sharada.sinor.ru/library/

Достаточно оперативный и со вкусом оформленный сайт, быстро откликающийся на все появляющиеся новинки в программном обеспечении.



VB Guru

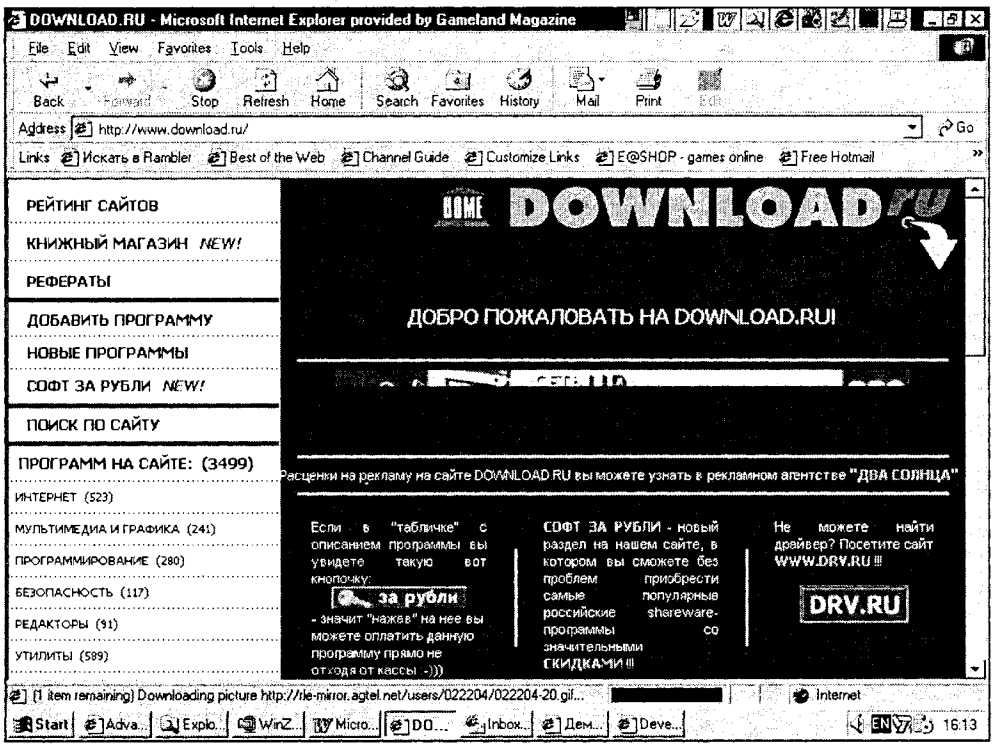
Еще один сайт, посвященный Visual Basic, — настоящий Гуру для программистов!

http://codeguru.earthweb.com/vb/

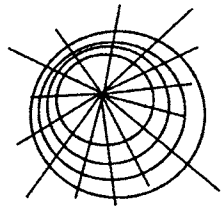
Сайт бесплатного программного обеспечения

www.download.ru

Содержит невообразимое количество многоразличного "софта" на все случаи жизни. В последнее время на этом сайте лидирует Интернет-ориентированное программное обеспечение в основном под Windows и Linux.



ГЛАВА 2



Что минимально нужно знать разработчику о JavaBean

В чем суть архитектуры JavaBean

Для тех, кто уже программирует на Delphi или Visual Basic, суть та же: написание повторно используемых компонентов и сборка из них готовых программ, как из первичных строительных блоков. Совокупность компонентов, выбираемых из специального контейнера (обычно, лента меню в RAD-системах), может включать набор простейших кнопок, текстовых скроллинговых (имеющих полосы прокрутки) полей и окон со списками или более сложные компоненты, такие как календарь или даже электронная таблица (рис. 2.1, 2.2).

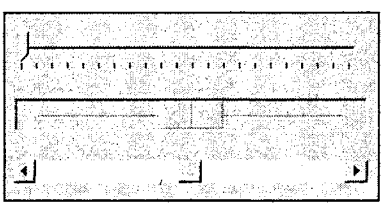


Рис. 2.1. Компонент "Слайдер"

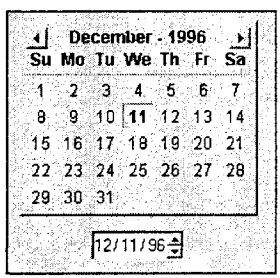


Рис. 2.2. Компонент "Календарь"

В традиционных языках программирования имеются специальные средства для создания таких компонентов, например, инструменты Visual Basic Extension (VBX), "языково-независимые" элементы OLE Custom Control (OCX), специальные средства в ОС Unix — toolkits и widgets. Однако зерна черного кофе JavaBeans — в просторечии "бинсы" или даже просто Бобы, имеют мост к элементам ActiveX и могут вкладывать себя в эту технологию. JavaBeans Bridge for ActiveX обеспечивает пользователей всех продуктов, использующих технологию ActiveX, например, таких как Word, Visual Basic,

Delphi и др., выступающих как полноценные OLE/COM/ActiveX-контейнеры, возможностью встраивать и использовать переносимые JavaBean-компоненты точно так же, как если бы они были родными ActiveX-элементами. Специальная графическая утилита записывает в OLE-формате библиотечную и регистровую информацию о выбранном JavaBean, т. е. как бы вводит его в семью OCX-элементов. Bean может действовать как сервер OLE/COM/ActiveX, т. е. любой его метод может быть вызван из Visual Basic или Delphi. В то же время и сам Bean способен реагировать на события, поступающие из OLE/COM/ActiveX-контейнера. Например, он может обрабатывать события, инициированные в языке сценариев Visual Basic Script (VBScript). Для программистов это означает, что они смогут пополнять свои большие, написанные, к примеру, на Visual Basic или Delphi приложения, модулями на Java, если они оформлены как JavaBean. Программист может вообще не задумываться, каким образом этот модуль сделан, просто он будет использовать в своих целях его свойства и методы. Парадигма программирования расширяется еще и тем, что JavaBean имеют много других замечательных особенностей.

Рассмотрим их по порядку. Начнем с того, что JavaBean является надстройкой над традиционной Java-технологией, хотя ни в чем ей не противоречит и использует только Java-конструкции, синтаксис и семантику Java-классов и интерфейсов. Однако JavaBean — это не просто библиотеки классов, — компоненты могут иметь изменяемые свойства, наборы методов и порождать события. Событийная модель в компонентах очень изощренная. Например, изменение свойств какого-либо компонента может служить источником события и об этом извещаются другие компоненты, в том случае, если они зарегистрировались на обработку этого события в качестве подписчиков. Само извещение иногда называют *распространением события* и при этом у объектов подписчиков вызывается специальный метод обработки события. Самое интересное заключается в том, что всегда можно выяснить, что может и чего не может тот или иной компонент. Компонент обязан *афишировать* свои характеристики, в которые входят:

- методы, которые можно вызывать;
- свойства, которые можно изменять;
- события, которые можно обрабатывать.

Выяснение данных характеристик со стороны других компонентов называется *интроспекцией*. В архитектуре Java эти методы были реализованы изначально и заключены в базовый класс `java.lang.reflect`. Тем не менее в архитектуре JavaBean существуют дополнительные классы `BeanInfo` и `Introspector`.

Интересно отметить, что немного времени спустя после опубликования стандартов на спецификации JavaBean сразу стали появляться системы программирования на Java (например, Visual Age фирмы IBM), которые предлагают разработчикам графические средства программирования для построения собственных событийных моделей и управления взаимодействием

компонентов на основе JavaBean. Это очень важно, если учитывать еще и то обстоятельство, что хотя в базовой концепции JavaBean описывается только локальное взаимодействие компонентов в пределах одной виртуальной Java-машины, на самом деле подразумевается, что эти компоненты могут находиться где угодно и что они могут взаимодействовать друг с другом по протоколу RMI или CORBA. Графическое программирование применяется для того, чтобы минимизировать, а в идеале вообще исключить ручное написание кода и включение его в программу при помощи текстового редактора. Вместо этого используются чисто графические наглядные средства, которые связывают схематические изображения компонентов и определяют их реакцию на то или иное событие с помощью задания свойств или вызова нужных методов. На рис. 2.3 изображено, как при помощи графических средств программируются функции компонента "Калькулятор". Выбор событий и действий над ними осуществляется из предлагаемого списка. Исходный код на Java генерируется автоматически.

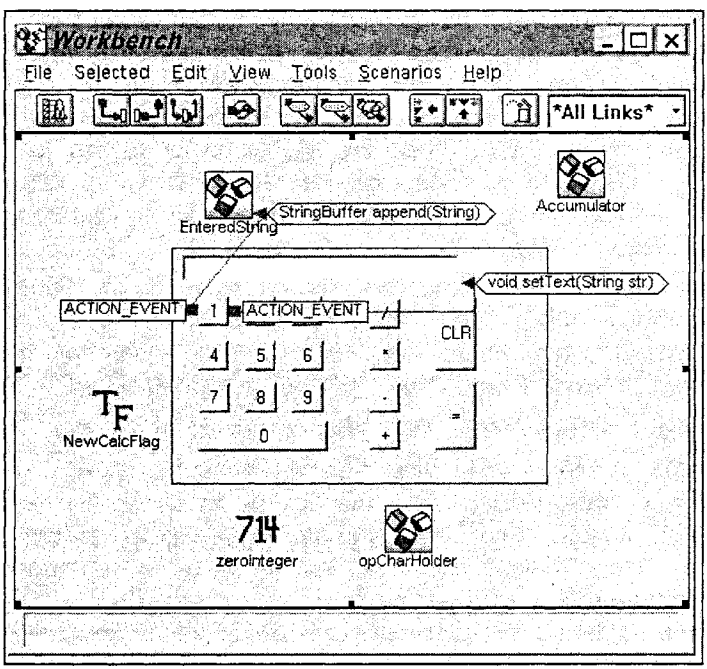


Рис. 2.3. Графическое программирование функций калькулятора

Однако одним визуальным программированием, естественно, не исчерпываются возможности JavaBean. Это всего лишь иллюстрация их вероятных областей применения. Как уже говорилось, серверы баз данных могут выступать в качестве контейнеров JavaBean, предоставляя пользователям свои объекты с афишируемыми функциями, тесно связанными с данными и спо-

собами их извлечения и обработки. Представляете, какие возможности открываются программистам и проектировщикам логических моделей баз данных и алгоритмов, связанных с данными? Конкретным же пользователям можно предложить очень простой интуитивный интерфейс для самых невероятных запросов, включая и аналитические. В базах данных, благодаря JavaBeans, теперь могут сохраняться разнообразные данные, в том числе и самых непредсказуемых типов. В этом смысле на первый план выступает проблема подготовки специалистов, способных усвоить и найти применение этим поистине безграничным возможностям. Проектировщики должны начать мыслить в совершенно иных категориях и других, непривычных системах координат. Это очень трудно. Поэтому главная роль в будущем будет принадлежать системному аналитику, системному проектировщику, а не программисту. Это хорошо понимали еще в 60-е годы создатели первых операционных систем, которые называли себя системными архитекторами. В книге "Как создаются и разрабатываются большие системы" одного из создателей памятника человеческому гению — операционной системы IBM OS/360 — Брукса-младшего приводится замечательное высказывание. Автор говорит о том, что он дал бы миллион долларов тому, кто изобрел *идею* часов — циферблат и стрелки, заключенные в металлический корпус с прикрепленным к нему ремешком, и всего только 100 тысяч долларов тому, кто привел этот механизм в действие, т. е. изготовил и соединил вместе все шестеренки, пружинки, оси и т. д. Программирование никогда не должно становиться самоцелью, это всего лишь средство. Компонентная архитектура JavaBeans предполагает именно такой подход к программированию.

Идея часов — это пример истинной компонентной модели с тремя основными компонентами: корпус, циферблат и стрелки, и она доказывает, что никогда нечто не возникает из ничего. Все мы, в той или иной мере являемся наследниками идей, изложенных в книгах Брукса, Дейкстры, Йодана.

Какие еще возможности предлагает компонентная архитектура JavaBeans?

После того как компонент разработан, он может быть сохранен на компьютере в точности так же, как и обычные файлы. Java помнит структуру объекта и вы, при желании, всегда можете воспользоваться интерфейсом `java.io.Serializable`, чтобы получить стандартную форму графа объектов, или преобразовать компонент при помощи интерфейса `Externalizable`, когда надо представить компонент в качестве элемента другой объектной среды, например, ActiveX. Естественно, что слишком большие компоненты можно упаковать и это даже рекомендуется для снижения трафика сети, когда вы работаете в распределенной системе. Для этого необходимо использовать стандартный Java-архиватор, который создает файлы с расширением .jar. Java-архивы могут содержать классы, изображения, справочную информацию, и к каждому архиву может быть прикреплен специальный файл *manifest*, описывающий содержание архива и даже связи между компонентами.

JavaBean можно создавать при помощи пакета BDK или из любой RAD-системы, поддерживающей эту технологию, например, Sybase PowerJ или Borland JBuilder. В сущности, эти системы просто облегчают работу кодировщику, предлагая ему каркасы классов различных типов, в которые он вкладывает свои коды или использует готовые компоненты JavaBean. Точно так же, как и обычные программы и апплеты, новый Bean можно создать по образцу. Нужно просто следовать определенным концептуальным правилам, хорошо изложенным в спецификации JavaBean.

В настоящее время доступна для опробования новая версия Enterprise JavaBeans, в которой детально проработаны соотношения компонент/контейнер. Теперь в единую объектную среду можно инкапсулировать совокупность компонентов, объединенных в контейнере, а с другой стороны — сам контейнер может выступать в виде родительской среды, скрывая от компонентов особенности внешней, возможно, даже и чуждой им среды и, предоставляя им для функционирования, свой контекст. Заслуживает также внимания и инициатива фирмы IBM/Lotus, предложившей фирме Sun обогатить имеющийся интерфейс JavaBean средствами для эффективного обмена данными между компонентами, получившими название InfoBus. Ключевым понятием в этой спецификации, уже вошедшей во все новые реализации JDK, является *информационная шина*. Эта шина описывается классом `InfoBus`. Компоненты подключаются к ней и выставляют на нее данные, в том числе и структурированные, которыми они хотят обменяться с другими компонентами. Обмен происходит в асинхронном режиме и не накладывает никаких ограничений. Доступ к данным осуществляется по их именам. Методы класса `InfoBus` порождают экземпляры класса, т. е. информационные шины обеспечивают подключение к ним компонентов и обрабатывают события, связанные с обменом данными.

Теперь рассмотрим все вышеописанные декларации более конкретно и подробно.

Что такое Bean (Боб)?

Любой объектный компонент можно рассматривать как фрагмент программного кода, обычно воплощаемого в классе. Чтобы создать VB-компонент, программист должен написать весьма сложный фрагмент программного кода и неукоснительно следовать условиям для правильного определения свойств, методов и событий, присущих этому компоненту. Поскольку Delphi является вторым поколением визуальных средств программирования, то в нем создавать компоненты стало намного проще. Что же касается Java, то Bean ("зерна кофе" (beans) — основные компоненты, из которых строится программа) — это не что иное, как класс. Вам нет необходимости разрабатывать дополнительный код или использовать специальные языковые рас-

ширения, чтобы выполнить что-то в виде Bean (Боба). Единственная вещь, которую вы должны сделать, — немного изменить способ присвоения имен вашим методам. Этот способ именования должен быть понятен средству разработки прикладных программ (RAD-системе), чтобы легко можно было уже по имени определять, что это — свойство, событие или вызываемый на выполнение метод.

В документации по Java соглашение об именах ошибочно именуется "design pattern", хотя это вовсе никакие ни шаблоны для разработки, а просто соглашение об именах и больше ничего. При проектировании компонентов Bean, обычно принимается следующее:

1. Для свойства с именем `xxx` обычно создаются два метода: **`getXxx()`** и **`setXxx()`**. По этим методам можно установить или получить свойства объекта.
2. Для логического выбора используется нечто подобное, но с именем "`is`" вместо "`get`."
3. Для событий используется ключевое слово "`listener`", например: **`addFooBarListener(FooBarListener)`** или **`removeFooBarListener(FooBarListener)`**, чтобы обрабатывать событие **`FooBarEvent`**. Большинство событий и встроенных обработчиков, как правило, устраивают разработчиков, но не возбраняется попробовать написать свой собственный `listener` для обработки какого-нибудь специфического события.

Попробуем написать свой собственный компонент Bean. За основу возьмем стандартный класс Java, который задает некий прямоугольник, и переопределим некоторые его свойства. Код на Java для `SimpleBean` будет простейший:

```
import java.awt.*;
import java.io.Serializable;

public class SimpleBean extends Canvas
                     implements Serializable{

  //Constructor sets inherited properties
  public SimpleBean(){
   setSize(60,40);
   setBackground(Color.red);
       }
}
```

Последовательность шагов, которую нам надо выполнить для создания полноценного компонента Bean, будет такая:

1. Компиляция разработанного кода

   ```
   javac SimpleBean.java
   ```

   В результате выполнения этого оператора образуется файл SimpleBean.class.

2. Подготовка манифест-файла.

   Дело в том, что при пакетировании готового Bean в jar-файл с использованием стандартного Java-архиватора jar требуется включить в архив специальный текстовый файл, называемый *манифест-файлом*, который описывает содержимое архива, т. е. в общем виде из чего архив состоит. Программные контейнеры, которые впоследствии будут использовать этот боб, извлекая его из jar-файла, всегда требуют, чтобы манифест-файл был. Подробнее о структуре манифест-файла можно прочитать в описании jar-программы для Java. Она всегда входит в бесплатную поставку JDK любой версии. В случае `SimpleBean` манифест-файл будет выглядеть следующим образом:

   ```
   Name: SimpleBean.class
   Java-Bean: True
   ```

   Эти две строки надо запомнить в текстовом файле с произвольным именем, например, manifest.tmp.

3. Далее надо упаковать созданный Bean в архив и обязательно включить в него манифест-файл. Это делается по команде:

   ```
   jar cfm SimpleBean.jar manifest.tmp SimpleBean.class
   ```

   Если Bean состоит из нескольких производных классов и образует Java-пакет, например, с именем `bangbean`, то его манифест-файл будет выглядеть следующим образом:

   ```
   Manifest-Version: 1.0
   Name: bangbean/BangBean.class
   Java-Bean: True
   ```

   При этом манифест-файл помещается в корень, и строка для упаковки компонента Bean со всеми его классами, размещающимися в каталоге c:\temp\bangbean, будет такая:

   ```
   jar cfvm BangBean.jar manifest.txt -C bangbean/*.class
   ```

   При этом после отработки команды архиватор jar выдаст вам следующую информацию:

   ```
   added manifest
   adding: bangbean/BangBean$ML.class (in=1556) (out=848) (deflated 45%)
   adding: bangbean/BangBean$MML.class (in=668) (out=417) (deflated 37%)
   adding: bangbean/BangBean.class (in=2740) (out=1334) (deflated 51%)
   ```

После этого считаем, что вы создали вполне работоспособный Bean, который можно загружать куда угодно и работать с ним так же, как и с элементом управления ActiveX. Можно переопределять его свойства, вызывать его методы, заставить его реагировать на определенные события и т. д. Заметь-

те, что Bean — это гораздо более гибкий объект, чем ActiveX, потому что это всего лишь класс, а не специализированная программа или библиотека. Данный класс написан на переносимом языке программирования, поэтому наш Bean можно даже включать в Delphi, в Visual Basic так же, как мы включаем в них купленные нами или самостоятельно разработанные управляющие элементы ActiveX. В дальнейшем вы увидите, как можно это сделать, а пока поработаем с Bean SimpleBean в каком-нибудь IDE, например, Visual Age. Из Visual Age загрузим BeanBox-контейнер, входящий в набор примеров Sun BDK Examples. Конечно, это можно сделать и непосредственно из Sun BDK (Bean Development Kit — бесплатная поставка), но некоторые его просто не имеют, хотя почти в любой RAD-системе, и в Java Builder, и в Visual Cafe, и в Visual Age, есть примеры работы с BDK. После активизации BeanBox-контейнера (рис. 2.4) он автоматически загружает в свой левый фрейм все имеющиеся в каталоге jars уже готовые компоненты Bean. Их можно перетаскивать на рабочую панель и использовать. Если вы хотите загрузить новый Bean, надо просто перейти в меню **File→LoadJar** и указать местоположение архива с этим компонентом. После загрузки имя его появится в левом фрейме.

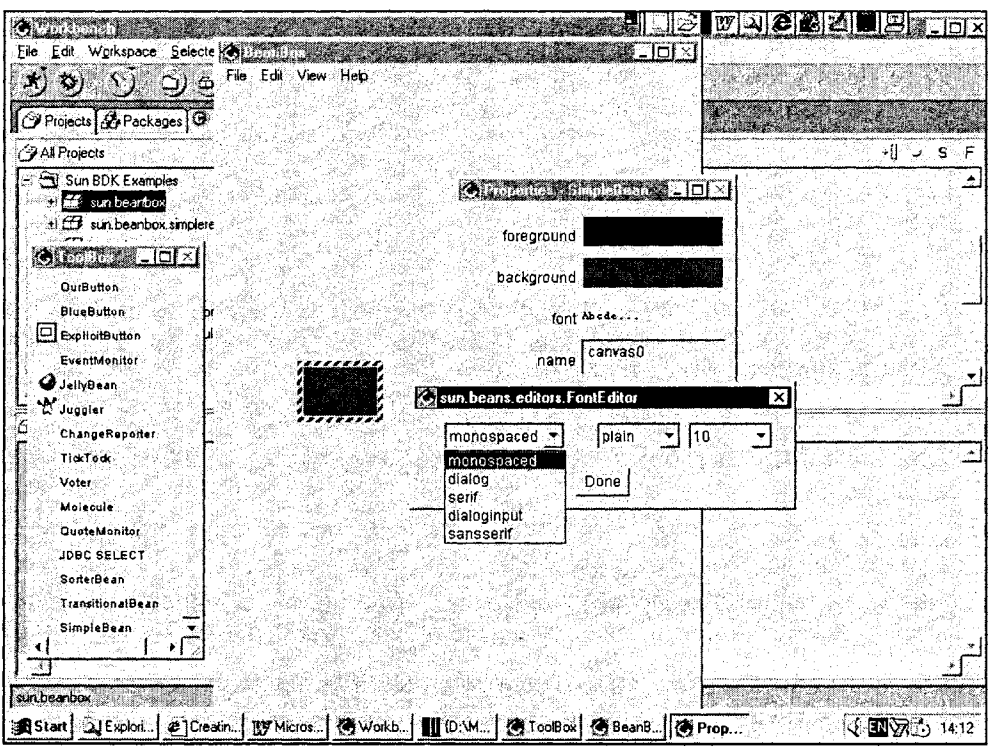


Рис. 2.4. Загрузка `SimpleBean` в контейнер `sun.beanbox` из среды Visual Age

На первый взгляд во всем этом нет ничего необычного, однако, посмотрим, что с этим Bean можно сделать еще? Для облегчения работ по дальнейшему совершенствованию и модификации Bean можно подготовить два Make-файла: один — для Unix, а другой — для Windows NT.

```
# gnumake file
CLASSFILES= SimpleBean.class
JARFILE= SimpleBean.jar
all: $(JARFILE)
# Создадим JAR-файл с соответствующим  манифест-файлом.
$(JARFILE): $(CLASSFILES) $(DATAFILES)
        echo "Name: SimpleBean.class" >> manifest.tmp
        echo "Java-Bean: True" >> manifest.tmp
        jar cfm $(JARFILE) manifest.tmp *.class
        @/bin/rm manifest.tmp
# Скомпилируем исходный файл
%.class: %.java
        export CLASSPATH; CLASSPATH=.; \
        javac $<
# очистка
clean:
        /bin/rm -f *.class
        /bin/rm -f $(JARFILE)
```

Теперь подготовим версию для Windows:

```
# nmake file
CLASSFILES= simplebean.class
JARFILE= simplebean.jar
all: $(JARFILE)
# Создадим JAR-файл с соответствующим  манифест-файлом.
$(JARFILE): $(CLASSFILES) $(DATAFILES)
        jar cfm $(JARFILE) << manifest.tmp *.class
Name: SimpleBean.class
Java-Bean: True
<<
.SUFFIXES: .java .class
{sunw\demo\simple}.java{sunw\demo\simple}.class :
        set CLASSPATH=.
        javac $<
clean:
        -del sunw\demo\simple\*.class
        -del $(JARFILE)
```

Инспекция свойств и событий

Поскольку Bean-компонент имеет программируемые свойства, которые в нашем случае он наследует из класса `Canvas` программы контейнера, работающей с Bean, всегда есть возможность узнать обо всех возможных свойствах и при желании переустановить их через специальное диалоговое окно, так называемый *редактор свойств*. Например, в программе-контейнере sun.beanbox, доступной в исходном коде по команде **Edit→Report**, можно узнать обо всех свойствах и обрабатываемых событиях рассматриваемого Bean. Отчет выдается на консоль в следующей форме:

```
CLASS: SimpleBean

Attributes:
    Hidden-state      null

H => Hidden
E => Expert
[ => Indexed Property
* => Default Event or Property

Properties:
 E  enabled           boolean                isEnabled/setEnabled
    foreground        class java.awt.Color   getForeground/setForeground
H   visible           boolean                isVisible/setVisible
    background        class java.awt.Color   getBackground/setBackground
    font              class java.awt.Font    getFont/setFont
    name              class java.lang.String getName/setName

Event sets:
    mouse                   addMouseListener/removeMouseListener
        mouseExited
        mousePressed
        mouseReleased
        mouseClicked
        mouseEntered
    key                     addKeyListener/removeKeyListener
        keyReleased
        keyPressed
        keyTyped
    component               addComponentListener/removeComponentListener
        componentMoved
        componentResized
        componentHidden
        componentShown
```

```
    focus                   addFocusListener/removeFocusListener
        focusGained
        focusLost
    mouseMotion             addMouseMotionListener/removeMouseMotionListener
        mouseDragged
        mouseMoved

Methods:
    public boolean java.awt.Component.isVisible()
    public java.awt.Component java.awt.Component.locate(int,int)
    public void java.awt.Component.enable(boolean)
    public java.awt.Dimension java.awt.Component.getSize()
    public void java.awt.Component.addNotify()
    public java.awt.Image java.awt.Component.createImage(int,int)
    public boolean java.awt.Component.isFocusTraversable()
    public java.awt.Point java.awt.Component.location()
    public java.awt.Dimension java.awt.Component.size()
    public synchronized void java.awt.Component.add(java.awt.PopupMenu)
    public synchronized void
java.awt.Component.remove(java.awt.MenuComponent)
    public java.awt.Point java.awt.Component.getLocation()
    public boolean java.awt.Component.mouseDrag(java.awt.Event,int,int)
    public int
java.awt.Component.checkImage(java.awt.Image,java.awt.image.ImageObserver)
    public void java.awt.Component.printAll(java.awt.Graphics)
    public synchronized void
java.awt.Component.addComponentListener(java.awt.event.ComponentListener)
    public boolean
java.awt.Component.lostFocus(java.awt.Event,java.lang.Object)
    public java.awt.Rectangle java.awt.Component.getBounds()
    .
    .
    .
```

Ясно, что эту информацию можно использовать в программе-контейнере, что и делается. По сути Bean афиширует и документирует все свои свойства и методы. Beans связываются с другими Bean, посылая и получая уведомления о происходящих событиях. Чтобы увидеть, какие события `SimpleBean` может посылать, воспользуемся пунктом меню **Event** программы BeanBox. Отобразится список событий, сгруппированных по интерфейсам. Под каждой группой интерфейса будет показан список методов для данного события. Все они в нашем случае унаследованы от класса `Canvas` (холст). Извлечение информации из Bean обо всем, что с ним связано, называется *интроспекцией* — и это является одной из самых мощных особенностей объектно-ориентированного программирования на Java. Интерфейс интроспекции либо использует соглашение о именах, описанное в документации по Java (design patterns), либо берет информацию из специального класса `BeanInfo`,

который может быть связан с каждым разрабатываемым JavaBean и должен быть обеспечен его разработчиком. Существует специальный класс `Introspector`. По заданному компоненту он конструирует объект класса `BeanInfo`. Действует Introspector следующим образом. Сначала он пытается найти класс, имя которого получается из имени класса компонента путем добавления текстовой приставки "`BeanInfo`". Если такой класс находится, соответствующая информация используется при конструировании результирующего объекта `BeanInfo`. В противном случае Introspector использует механизм рефлексии, т. е. соглашение об именовании (design patterns), и анализирует имена и типы параметров public-методов класса компонента и тех, которые он наследует. К сожалению, данный предмет обсуждения слишком велик и выходит за рамки настоящей книги, однако, как и все, связанное с программируемыми объектами в Java, особенности программирования Bean имеют огромное количество примеров в Интернете, хорошо описаны в фирменной документации по Java, не говоря уже о таких канонических книгах, как "Thinking in Java" ("Философия Java") легендарного Б. Эккеля (B. Eckel) и др.

Использование компонентов Bean в качестве ActiveX-элементов

Преимуществом открытой архитектуры JavaBean является отрадный факт возможности миграции из приложений, основанных на применении ActiveX-решений, в кроссплатформенный мир чистого Java благодаря технологии, известной как JavaBeans Architecture Bridge for ActiveX.

Мост JavaBean-ActiveX

Преобразование JavaBean в ActiveX

Компоненты JavaBean могут быть использованы как встроенные объекты или компоненты Microsoft ActiveX внутри таких популярных приложений, как Microsoft Office, Internet Explorer и Visual Basic. Для этих целей можно применять стандартный Java Plug-in, бесплатно распространяемый фирмой JavaSoft, или воспользоваться возможностями JRE из JDK1.2.

Кратко о главном

Для того чтобы использовать JavaBean как встроенный объект, вы должны упаковать ваш JavaBean и затем применять его в качестве компонента ActiveX. Реестровый файл в Windows (текстовый файл) и библиотека описания типов (.tlb — в которой хранятся описания объектов, их свойства, доступные методы и т. д.), будут созданы специальной программой-упаков-

щиком, входящей в состав JRE. В реестре будет записан уникальный идентификатор объекта, путь к компоненту, описание компонента и путь к библиотеке описания типов. Файл TypeLib — это двоичный файл, который описывает все свойства компонента, обрабатываемые им события и исполняемые методы. Схематично процесс пакетирования изображен на рис. 2.5.

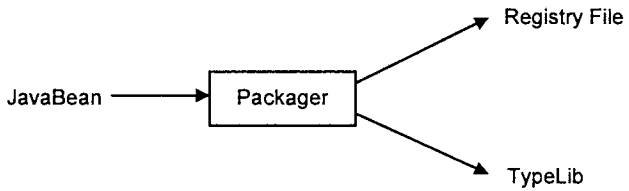


Рис. 2.5. Схема преобразования JavaBean в ActiveX (где Packager — программа-упаковщик, Registry File — реестровый файл)

Пакетирование и регистрация

Шаг 0. Старт упаковщика

Чтобы запустить упаковщик, выполните следующую командную строку из каталога, где у вас инсталлирован Java Plug-in, или непосредственно из \jdk1.2\jre\bin:

```
bin\java.exe -cp lib\rt.jar;lib\swingall.jar;lib\jaws.jar
sun.beans.ole.Packager
```

На экране монитора появится окно упаковщика и он проведет вас последовательно через 5 шагов.

Шаг 1. Указание местонахождения jar-файла

Укажите имя jar-файла, который содержит JavaBean, и нажмите кнопку **Next** (рис. 2.6).

Шаг 2. Выбор компонента JavaBean

Средство пакетирования показывает список всех компонентов JavaBean внутри jar-файла. Выберите необходимый вам JavaBean и нажмите кнопку **Next** (рис. 2.7).

Шаг 3. Присвоение имени ActiveX

Средство упаковки требует указания имени ActiveX для JavaBean. Это имя будет использоваться в среде ActiveX, чтобы обращаться к компоненту. Упаковщик предлагает имя по умолчанию (имя JavaBean минус префикс).

Так, если бы вы выбрали вместо `OrangeButton` в первом шаге — `sunw.demo.buttons.ExplicitButton`, то и предлагаемое имя было бы — `ExplicitButton` (рис. 2.8).

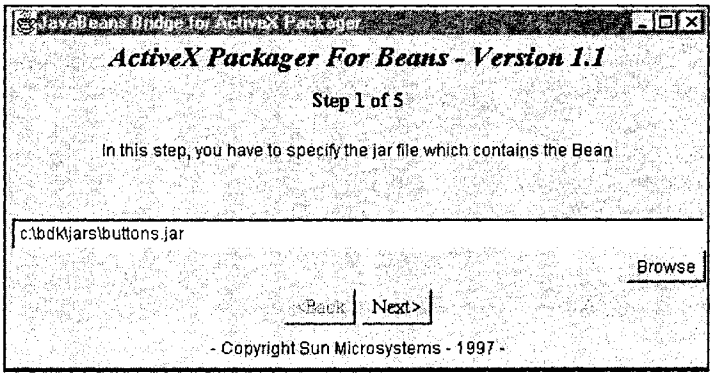


Рис. 2.6. Окно упаковщика

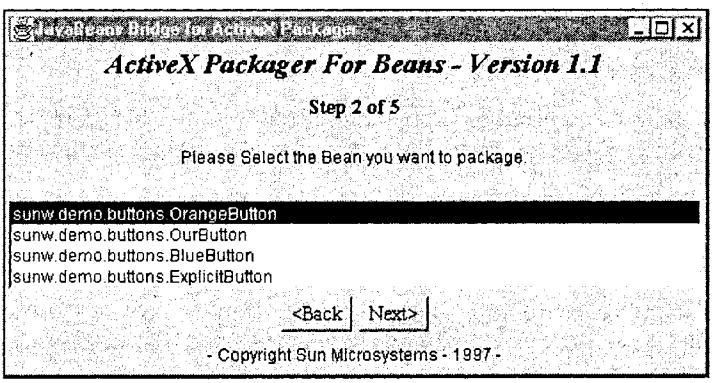


Рис. 2.7. Выбор из списка

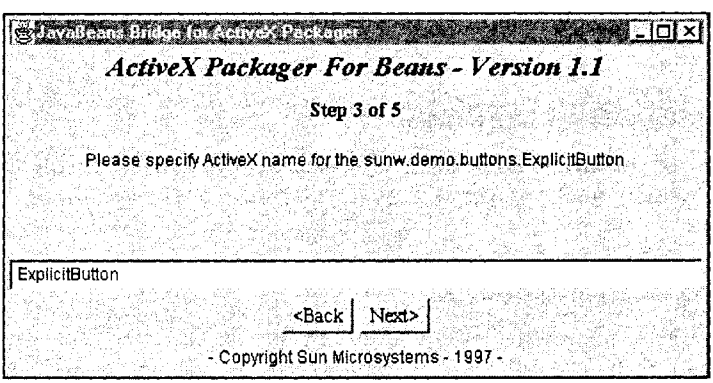


Рис. 2.8. Присвоение имени компоненту ActiveX

Шаг 4. Указание принимающего каталога

Упаковщик создает реестровый файл (.reg) и библиотеку описания типов (.tlb) в каталоге, который вы указали (рис. 2.9). Обратите внимание, что реестровый файл содержит информацию относительно расположения файла библиотеки классов, как еще иногда называют TypeLib. Если вы захотите переместить файл библиотеки классов, вам потребуется вручную модифицировать реестровый файл и зарегистрировать его снова. Местоположение beans.ocx (компонент Java Plug-in ActiveX) определено при инсталляции Java. Опять-таки, если вы перемещаете beans.ocx, вы должны вручную изменить этот файл.

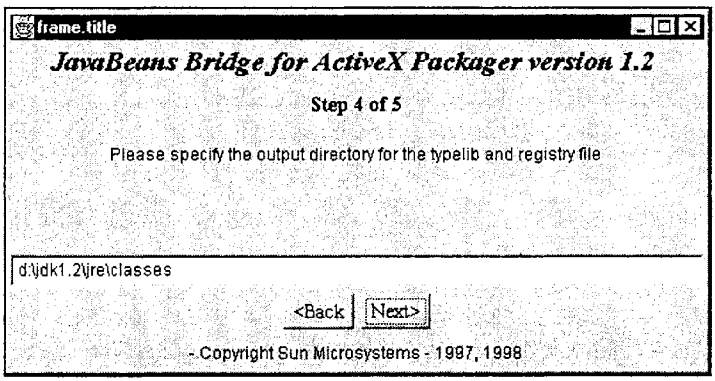


Рис. 2.9. Выбор принимающего каталога

Шаг 5. Начало генерации

Упаковщик автоматически регистрирует компонент в реестре Windows на вашей локальной машине. При выполнении этого шага компонент JavaBean будет автоматически доступен в любом ActiveX-контейнере (рис. 2.10).

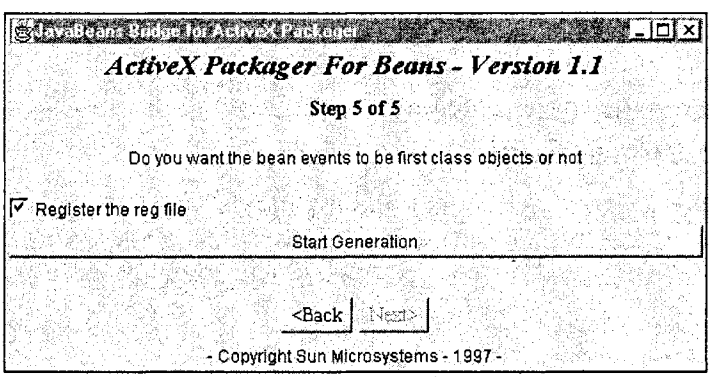


Рис. 2.10. Генерация ActiveX-компонента из JavaBean

Опции командной строки

Поскольку Microsoft обещала внедрить ActiveX и на Unix-платформы, Sun и JavaSoft на всякий случай зарезервировали для любителей работы с терминалов и алфавитно-цифровых дисплеев опции командной строки для упаковщика. Сам упаковщик может быть вызван в режиме командной строки с терминала по команде:

```
java sun.beans.ole.Packager
```

`-jar`	<путь к jar-файлу>
`-n`	<JavaBean-имя как ссылка в манифест-файле jar>
`-ax`	<ActiveX-имя, связанное с JavaBean>
`-o`	<приемный каталог (т. е. используемый упаковщиком для вывода (output))>

Вы можете указывать опции в любом произвольном порядке. Обязательной является опция `-jar`. Упаковщик использует некоторые значения по умолчанию.

`-n`	Упаковщик использует первый JavaBean, доступный в jar-файле. Эту опцию рекомендуется указывать, когда jar-файл содержит более чем один JavaBean
`-ax`	Упаковщик использует JavaBean-имя (минус префикс). К примеру, `sunw.demo.buttons.ExplicitButton` становится `ExplicitButton`
`-o`	Упаковщик генерирует файлы typelib и reg в текущем каталоге. Если желаете использовать другой каталог — укажите его здесь

Информацию о ходе работы упаковщик выводит на стандартный вывод, обычно в то окно, к котором он сам стартовал.

Подготовка к встраиванию Bean в программу на Visual Basic

Посмотрим, как можно встроить созданный нами ранее SimpleBean в программу на Visual Basic. После обработки SimpleBean.jar упаковщиком и сохранении результата в каталоге d:\mybook\examples, в указанном каталоге будет 2 файла:

SimpleBean.reg

SimpleBean.tlb

Поскольку файл SimpleBean.reg — текстовый, его можно распечатать. Ниже приведено его содержимое:

```
REGEDIT4
[HKEY_CLASSES_ROOT\SimpleBean.Bean]
@= "SimpleBean Bean Control"
[HKEY_CLASSES_ROOT\SimpleBean.Bean\CLSID]
@= "{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}"
[HKEY_CLASSES_ROOT\SimpleBean.Bean\CurVer]
@= "1"
[HKEY_CLASSES_ROOT\SimpleBean.Bean.1]
@= "SimpleBean Bean Control"
[HKEY_CLASSES_ROOT\SimpleBean.Bean.1\Insertable]
[HKEY_CLASSES_ROOT\SimpleBean.Bean.1\CLSID]
@= "{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}]
@= "SimpleBean Bean Control"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\InprocServer32]
@= "d:\\jdk1.2\\jre\\bin\\beans.ocx"
"ThreadingModel" = "Apartment"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\ToolboxBitmap32]
@= "d:\\jdk1.2\\jre\\bin\\beans.ocx,0"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\TypeLib]
@= "{F35B5A41-0613-11D4-A6AE-00C0F6F0023C}"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\ProgID]
@= "SimpleBean.Bean.1"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-
00C0F6F0023C}\VersionIndependentProgID]
@= "SimpleBean.Bean"

[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\JarFileName]
@="D:\\MYBOOK\\ActiveX_JavaBeans_Case\\mybook_examples\\Beans\\SimpleBean.jar"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\JavaClass]
@= "SimpleBean"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\InterfaceClass]
@= "sun/beans/ole/OleBeanInterface"

[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\Control]
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\Programmable]
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\Insertable]
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\MiscStatus]
@= "0"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\MiscStatus\1]
@= "18833"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\DefaultIcon]
@= "d:\\mybook\\examples\\awt.ico"

[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\Version]
@= "1.0"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\DataFormats\GetSet\0]
@= "2,1,16,1"
```

```
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\DataFormats\GetSet\1]
@= "3,1,32,1"

[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\DataFormats\GetSet\2]
@= "14,1,64,1"

[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\DataFormats\GetSet\3]
@= "1,1,1,1"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\verb\0]
@= "&Edit,0,2"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\verb\-1]
@= "Show,0,0"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\verb\-2]
@= "Open,0,0"
[HKEY_CLASSES_ROOT\CLSID\{F35B5A40-0613-11D4-A6AE-00C0F6F0023C}\verb\-3]
@= "Hide,0,1"

[HKEY_CLASSES_ROOT\TypeLib\{F35B5A41-0613-11D4-A6AE-00C0F6F0023C}]
@= "SimpleBean Bean Control Type Library"
[HKEY_CLASSES_ROOT\TypeLib\{F35B5A41-0613-11D4-A6AE-00C0F6F0023C}\1.0]
@= "SimpleBean Bean Control "
[HKEY_CLASSES_ROOT\TypeLib\{F35B5A41-0613-11D4-A6AE-00C0F6F0023C}\1.0\0\win32]
@= "d:\\mybook\\examples\\SimpleBean.tlb"
[HKEY_CLASSES_ROOT\TypeLib\{F35B5A41-0613-11D4-A6AE-00C0F6F0023C}\1.0\FLAGS]
@= "2"
[HKEY_CLASSES_ROOT\TypeLib\{F35B5A41-0613-11D4-A6AE-00C0F6F0023C}\1.0\HELPDIR]
@= "d:\\mybook\\examples"
```

Как видите, все готово для регистрации, стоит только щелкнуть мышью по файлу. Но этого делать не надо, потому что упаковщик уже проделал это. Компонент `SimpleBean` уже зарегистрирован на локальной машине (если вы хотите зарегистрировать его на удаленной и использовать через DCOM, вам потребуется вручную скорректировать этот файл). Теперь давайте использовать созданный на Java ActiveX-компонент в практических целях. Запустим его в Бэйсике!

Импортирование JavaBean в Visual Basic

Из Visual Basic выберем в меню **Project** пункт **Components** и добавим компонент `SimpleBean`, как показано ниже (рис. 2.11).

После того как мы его выбрали, он мгновенно появится в левом фрейме окна разработки, где VB обычно показывает все имеющиеся в его распоряжении элементы управления ActiveX (ActiveX Controls). Вот и наш появился там в виде пиктограммы с изображением белого листочка. Перетащим его в форму. Все OK! Мы можем работать с компонентом. Задавать ему свойства в редакторе свойств и т. д. Программа Visual Basic даже не чувствует и не может почувствовать, что это не родной компонент ActiveX и что он был создан на Java. В этом и заключается прелесть кросс-платформенных технологий (рис. 2.12).

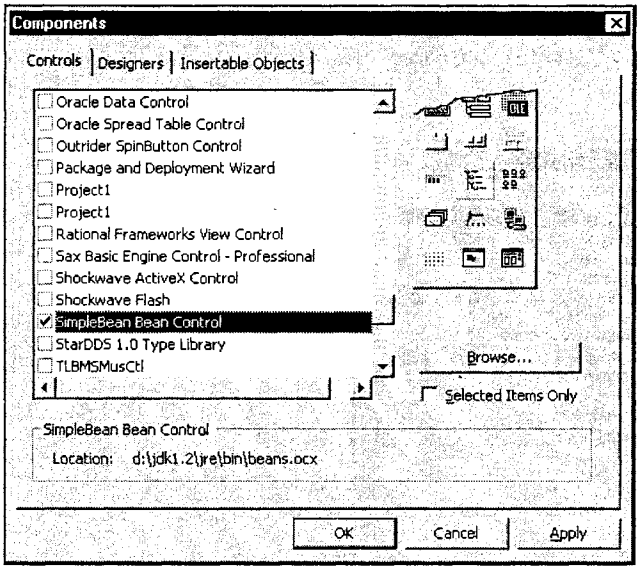


Рис. 2.11. Выбор компонента `SimpleBean` в Visual Basic

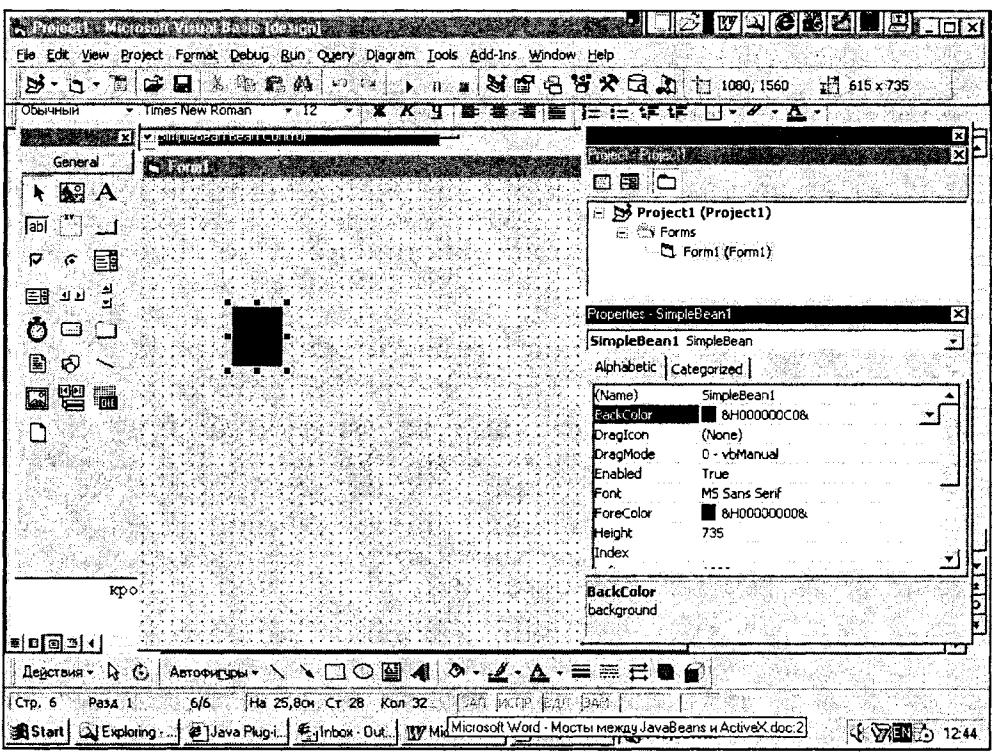


Рис. 2.12. Использование `SimpleBean` в VB IDE в виде ActiveX-компонента

На самом деле, если бы наш компонент не был таким простым, мы могли бы перепрограммировать его методы, которые также показываются в окне (рис. 2.13).

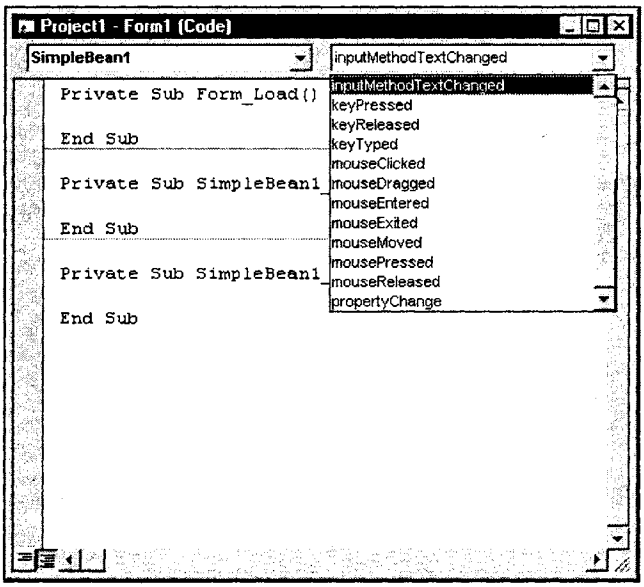


Рис. 2.13. Список доступных методов `SimpleBean`

Но дело вовсе не в том, чтобы до бесконечности усложнять пример. Главное — это понять принцип создания таких компонентов и затем использовать их в своей работе. Две технологии Microsoft и Sun здесь не враждуют, а взаимно обогащаются. Впрочем, для наиболее любознательных читателей, я, все-таки, не откажу себе в удовольствии привести пример из технического описания Java ActiveX Bridge, в котором показывается возможность взаимодействия двух Beans, находящихся в одном ActiveX-контейнере.

Для этого потребуется скрипт (сценарий) для первоначального соединения. Как только это будет выполнено, два компонента связываются непосредственно друг с другом без использования COM. Это позволяет JavaBean-компоненту получать и обрабатывать события других JavaBean-компонентов, выполняющихся в том же самом контейнере без дополнительных усилий взаимодействия со всеми уровнями COM.

Рассмотрим компонент JavaBean, который выполняет интерфейс `java.awt.event.Action-Listener`, чтобы обработать событие, связанное с нажатием кнопки. Чтобы сделать это, JavaBean может непосредственно выполнить интерфейс `listener`, или, что гораздо проще, обеспечить класс `adaptor`, который делает это за него. В примере используются дополнительные возможности JDK1.1 — вложенные классы.

```
Class JavaBeansExample {
.... Некоторый код....
/*
* Небольшой класс adaptor, который использует ActionListener
*
*/
public java.awt.event.ActionListener getActionListener() {
return new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {
Dialog myDialog = new Dialog(new Frame(), "Event Received");
myDialog.reshape(200,200,400,400);
myDialog.show();
}
};
}
.... Некоторый код....
```

Вызов метода `getActionListener` в этом JavaBean возвращает объект, который осуществляет интерфейс `ActionListener`. Реализация интерфейса `actionPerformed` позволяет создать новое диалоговое окно, чтобы сообщить пользователю, что событие `ActionEvent` было получено. Экземпляр JavaBean `ExplicitButton`, который поставляется как часть BDK, в исходных текстах использует интерфейс `ActionListener`. Вы можете привязать реализацию интерфейса первым JavaBeans-компонентом к экземпляру `ExplicitButton` в диалоговом окне, когда пользователь нажимает на кнопку. Соединение между компонентами станет возможным, когда будет загружена форма Visual Basic:

```
Private Sub Form_Load()
ExplicitButton1.addActionListener(JavaBeans1.getActionListener)
End Sub
```

Если компонент `JavaBeansExample` непосредственно выполняет `java.awt.event.ActionListener`, то код должен выглядеть так:

```
Class JavaBeansExample implements java.awt.event.ActionListener {
... Некоторый код ...
public void actionPerformed(java.awt.event.ActionEvent e) {
Dialog myDialog = new Dialog(new Frame(), "Event Received");
myDialog.reshape(200,200,400,400);
myDialog.show();

... Некоторый код...
}
```

И метод соединения такой:

```
Private Sub Form_Load()
ExplicitButton1.addActionListener(JavaBeans1.Object)
End Sub
```

Любой подход допустим. Когда пользователь нажимает кнопку (JavaBeans ExplicitButton), событие перехватывается компонентом JavaBeanExample, который затем отображает диалоговое окно. Эти прямые соединения не ограничены событиями. Объект, созданный одним компонентом JavaBean, может быть передан как параметр метода для другого JavaBean. Обратите внимание, что первоначальный объект передается не как копия.

Прямое соединение в Internet Explorer

Такое же прямое соединение может быть также установлено при выполнении MS IE. Следующий код, который выполняется, когда MS IE обрабатывает событие Window onLoad event (все компоненты загружаются на страницу), демонстрирует то же самое взаимодействие между двумя компонентами, как и в предыдущем примере.

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub  window_onLoad()
 x = test1.getActionListener
 button1.addActionListener x
end sub
-->
</SCRIPT>
```

OLE Automation Server

Существует возможность сделать так, чтобы JavaBeans-компонент действовал как OLE Automation Server (без пользовательского интерфейса JavaBeans Bridge for ActiveX имеет встроенную поддержку этой возможности и при этом ровным счетом ничего не приходится менять в самом компоненте).

Ниже представлен один пример, где уже известный нам SimpleBean действует как OLE-сервер, а клиент Visual Basic имеет возможность вызывать его методы. Поскольку у нас нет собственных методов, а есть только унаследованные из класса Canvas, вызовем хотя бы стандартный метод получения имени класса и распечатаем его в форме.

```
Rem // Загрузим JavaBeans как OLE Automation Server
Rem // "SimpleBean.Bean.1" это ProgID JavaBean-компонента
Dim SimpleBean As Object
Set SimpleBean = CreateObject("SimpleBean.bean.1")
```

```
Rem // Вызовем метод SimpleBean
MsgBox SimpleBean.getClass.toString
Rem // Освободим объект
Set SimpleBean = Nothing
```

Результат показан на рис. 2.14.

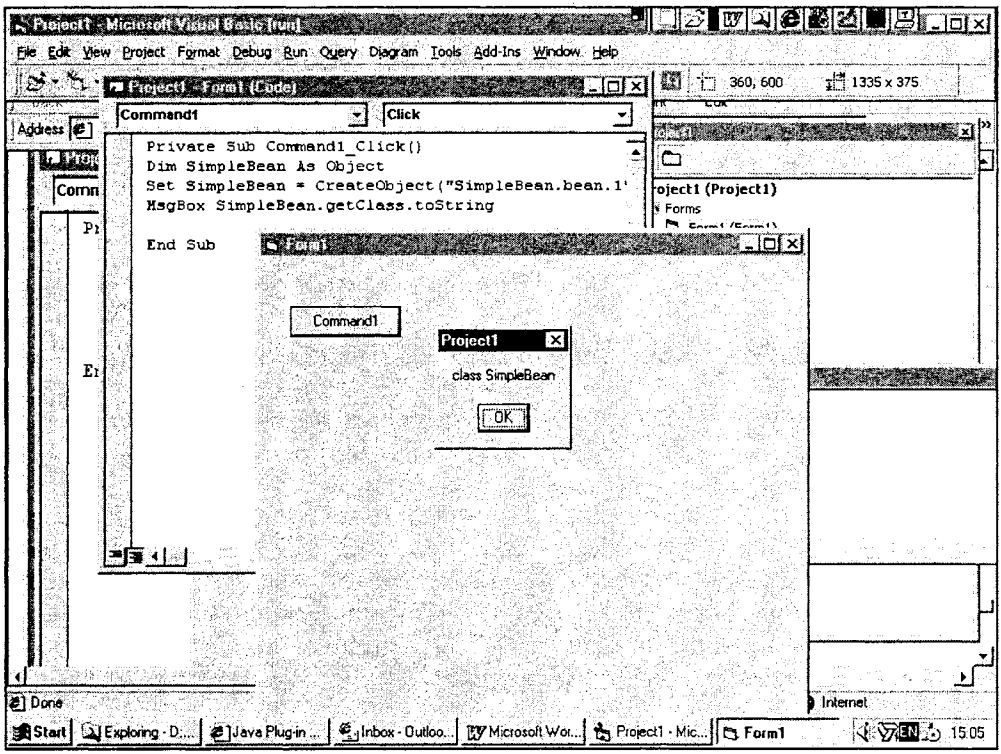
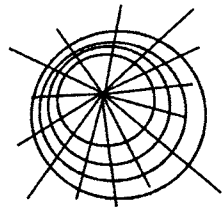


Рис. 2.14. Использование Java-компонента как OLE-сервера

ГЛАВА 3



CORBA и Enterprise JavaBeans — лидеры новейших технологий

Зачем нужна CORBA

Сразу оговоримся, что под CORBA мы будем понимать CORBA/IIOP, потому что сама спецификация CORBA не стандартизует сетевой протокол передачи данных между клиентом и сервером. Каждый производитель брокера объектных запросов, который, по сути и есть CORBA, может предложить собственный протокол транспортной службы передачи данных. Однако для обеспечения интероперабельности брокеров запросов от разных производителей OMG (Object Management Group — международная некоммерческая организация, в которую входят промышленные компании и компании, занимающиеся созданием программного обеспечения) разработал общий протокол GIOP. Его отображение в TCP/IP получило название протокола *Internet Inter ORB Protocol* (IIOP). Благодаря этому, любое приложение CORBA/IIOP должно работать и взаимодействовать с другими CORBA/IIOP-приложениями вне зависимости от платформ, производителей ПО и операционных систем.

CORBA (Common Object Request Broker Architecture) — это технология для мира распределенных информационных объектов, тесно взаимодействующих между собой в рамках управляющей ими программы, которая виртуально из этих объектов и состоит. Так вот, CORBA — это архитектура, которая с максимальными удобствами обеспечивает создание и функционирование программ, называемых *CORBA-приложениями.* В последнее время почти все RAD-системы: Delphi-5, Visual Age и др. включают в свой арсенал поддержку CORBA.

Впрочем, традиционные RAD-системы, подобные Visual Basic, не нуждаются в CORBA, потому что ориентируются на имеющийся в Microsoft собственный брокер объектных запросов, существующий под именем DCOM. В дальнейшем производителям программных продуктов для программистов станет все труднее и труднее поддерживать обе технологии. Я, признаться, и

сейчас не совсем понимаю — зачем Delphi 5 нужна технология CORBA, когда в той же фирме выпускается другой превосходный продукт JBuilder, для которого поддержка CORBA куда более естественна? Delphi-программисты могут успокаивать себя тем, что есть возможность работать и с CORBA, и с DCOM, хотя они чаще всего разрабатывают распределенные приложения именно для DCOM, а те из них, кто используют CORBA, предпочитают в дальнейшем иметь дело уже с JBuilder.

Конечно же, CORBA — более масштабируемый, более открытый и более грандиозный проект, чем DCOM. CORBA рассматривает уже существующие приложения, как некие метафоры, которые легко можно адаптировать и использовать, правильно и единожды описав их интерфейсы с помощью специального языка описаний IDL (Interface Definition Language — язык описания запросов), для которого существуют компиляторы в большинство современных языков программирования. CORBA является прообразом нашего объектного Будущего. В компьютерном мире совершенно нормальным считается факт, что программисты всех стран по миллиону раз переписывают заново один и тот же алгоритм, если он требует внесения некоторых небольших изменений. Зачем? Ведь в любом случае, безусловно, существует лучший вариант того фрагмента программного кода, который переписывается с самого начала людьми, как бы не знающими об этом. Почему, по сравнению с программистами, так далеко вперед ушли инженеры компьютерных систем? Да потому, что они уже давным-давно используют объектную технологию в виде неизменяемых конструкторских элементов ("чипов"). А программисты по старинке начинают с песочка: сначала просеим песочек, затем добавим в него глину, после этого изготовим кирпичи, а уж потом из тех кирпичей построим конструкцию. Кустарное производство. Что означают объектные технологии в программировании, и не только в программировании, а шире, — во всем компьютерном творчестве в эпоху безраздельного господства Сети. Мы вынуждены признать, что современное производство начисто лишено индивидуализма и признает только то, что может быть растиражировано. Наилучшим вариантом считается тот, который вы не создаете, а лишь настраиваете, компонуя его из различного набора объектов, описывая их свойства и задавая различные методы, выполняющие тот или иной вид работы. Выигрыш производительности получается колоссальный. Во всем царит единый принцип — повторное использование компонентов. Ваш труд, если вы являетесь создателем новых информационных объектов и одновременно клиентом системы объектной реализации, легко может превратиться в объект: процесс или программу, которые могут использовать другие клиенты и объекты с целью употребления ваших методов, если они содержат в себе нечто новое, вне зависимости от ваших желаний, местонахождения и времени. Конечно, было бы идеально, если бы повторно используемые объекты сами бы не являлись объектами, а их создатель обладал правами на плоды своего творчества. Но ведь невозможно представить себе, что бы создатель какого-нибудь объекта действовал как первичный демиург, т. е.

творил в вакууме, нет, он всего лишь наследует структуру какого-нибудь описания, какого-нибудь шаблона, класса или группы классов — пакетов. То что сейчас происходит с программными модулями, оформляемыми на универсальном языке программирования в JavaBeans или в ActiveX, то же, по существу, происходит и с авторскими текстами, превращающимися не в индивидуальные плоды творчества, а в один бесконечный и универсальный комментарий. Какой смысл заново описывать то, что уже однажды описано? Тем не менее это делается, и текст, проходя через множество интерпретирующих его объектов, постоянно шлифуется, до тех пор, пока не приобретает абсолютную форму, напоминающую сверкание алмаза — просто холодный и яркий изумительный свет, несущий в себе зерно истины, познанной через опыт. Если раньше Сеть еще носила на себе яркий отпечаток индивидуализма ее создателей, то теперь она все более и более превращается в некий Универсум, который по определению не терпит ничего, выходящего за его пределы. Все или почти все в Сети приобретает статус анонимности. Используются уже не полноценные имена, а всего лишь идентификаторы, не индивидуальные пароли, а сертификаты. Творцов не может быть слишком много. В конце концов единственным создателем всего становится сама Сеть. В этих условиях разговоры об авторских правах становятся просто смешны. Никто ничего не изобретает, он всего лишь наследует. Понятие полиморфизма также является неотъемлемой чертой систем объектных реализаций. Вы просто объявляете себя наследником каких-нибудь общеупотребительных классов: понятий, конструкций, образов, текстов и модифицируете их в соответствии со своими установками. Творец обретает себя всего лишь в извлечении новых смыслов из старых конструкций — форм и инструментов у него вполне достаточно. В наследуемом классе при этом ничего не изменяется, однако новый класс, являясь наследником старого, по воле своего создателя, содержит в себе переопределенные ключевые системные и смысловые блоки, так что они становятся полиморфными, т. е. ведут себя и представляются совершенно по-разному, в зависимости от того, кто и каким образом их востребовал. Примеры совершенно просты: вы включаете свой телевизор точно таким же способом, как утюг, электропечь или настольную лампу. Все предметы, относящиеся к классу "Прибор", содержат метод "включить", но в процессе включения, в зависимости от типа прибора (подкласса), единственный метод "включить" ведет себя полиморфно, т. е. вызывает совершенно различные электрические и блоковые функции: для утюга — одни, для телевизора — другие. Можно так переопределить классы и методы в телевизоре новейшей конструкции, что он будет вести себя совершенно по-разному, в зависимости от того, КТО его включил, т. е. телевизор поведет себя полиморфно. Более того, существуют программные средства (Java+XML), которые способны динамически модифицировать контент. Несмотря на то, что Java и XML сами по себе достаточно сложны, инструменты, основанные на них, для обычного пользователя чрезвычайно просты и приятны, ибо они уводят его от необходимости знать внутренние механизмы

информационного источника и приводят к дистанционному пульту управления им. Точно так же, как он сейчас управляет своим телевизором, не задумываясь о его техническом устройстве, в будущем, благодаря новым, все более совершенным технологиям, он будет способен управлять информацией, структура которой определяется пожеланиями клиента. С точки зрения систем объектной реализации, люди также представляют из себя объекты с программируемыми поведенческими свойствами. Они сами, будучи подключены к электронным системам, неоднократно дефинируют (переопределяют) свои информационные запросы к ресурсам, которые система запоминает и впоследствии использует. Всегда, как только человек вступает во взаимодействие с программируемыми объектами, он сам становится объектом, он сам становится программируемым. Для постмодернистской эпохи, завершением которой является Интернет, ценности в традиционном понимании не являются ценностями, ибо она манипулирует ими и строит из них структуры более высокого порядка. Плоть и кровь ушли из цивилизованных людей, осталось лишь электричество. Бессмысленно заявлять свои права на конструкции, которые существуют как узоры в калейдоскопе. Все становится виртуальным, все становится относительным, а точка отсчета — Сеть. Аспектом творчества становится поддержание жизни Сети, т. е. способствование перетеканию живого в электрическое. Современные люди уже не способны круто изменить парадигму своего существования. Творчество в эпоху Интернета представляет собой извлечение все новых и новых смыслов из известных вещей и постоянное обновление пространства контента.

Сила CORBA в том, что это самоописывающая и самодокументирующая система. Она была разработана для того, чтобы позволить создавать интеллектуальные компоненты, которые могут запрашивать информацию друг о друге и потом ее эффективно использовать. Число взаимодействующих объектов при этом не ограничено, способы существования и местонахождение объектов жестко не определены. Любой объект может быть востребован другим объектом или программой в произвольной точке Сети, и это очень сильно напоминает телепортацию.

Четыре главных элемента CORBA архитектуры показаны на рис. 3.1.

- **Брокер объектных запросов (ORB — Object Request Broker)** определяет механизмы взаимодействия объектов в разнородной сети. Посредник объектных запросов, ORB, или просто брокер, — это логическое ядро системы. Именно он позволяет объектам посылать запросы и получать ответы от других объектов в сети. ORB — это промежуточное ПО, которое устанавливает отношения между объектами в распределенной среде. Брокер по ссылке находит сервер, содержащий объект-адресат (object implementation — реализация объекта); активизирует его, доставляет к нему запрос; передает параметры и вызывает соответствующий метод, после чего возвращает результат клиенту. Роли клиента и сервера при этом могут меняться. Взаимодействие может быть установлено между множеством

объектов. OMG разработал специальный язык верхнего уровня для описания ORB и других компонентов CORBA — так называемый *IDL* (Interface Definition Language — язык описания интерфейсов). В нем нет присваиваний, обычных языковых конструкций, типа `if` или `while`, функций и логических переходов и т. д. IDL — это язык деклараций, он описывает родительские классы, события и методы. Каждый интерфейс определяет операции, которые могут быть вызваны клиентами, но не то — каким образом эти операции выполняются вне IDL и CORBA. В этом и прелесть — можно заставить работать старые программы, если правильно описать их интерфейсы и использовать компилятор с IDL, который имеет отображение в конкретный язык программирования. Важно понять, что имеется два типа интерфейсов: *динамические* и *статические*. Первые определяются на стадии разработки приложения и компилируются вместе с ним, вторые конструируются в момент исполнения, на работающем приложении.

- **Сервисы объектов (Object Services)** являются основными системными сервисами, которые разработчики используют для создания приложений.
- **Универсальные средства (Common Facilities)** являются также системными сервисами, но более высокого уровня, ориентированными на поддержку пользовательских приложений, таких как электронная почта, средства печати и т. д.
- **Прикладные объекты (Application Objects)** используются в конкретных прикладных задачах.

Так, все-таки, зачем нужна CORBA?

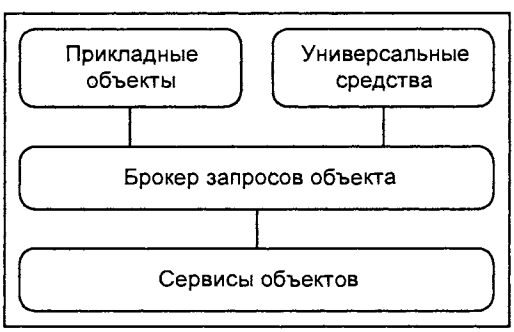


Рис. 3.1. Главные элементы CORBA

Распределенные приложения

CORBA-архитектура обеспечивает каркас для разработки и выполнения распределенных приложений. Но возникает новый вопрос — а зачем вообще нужно это распределение? Как только вы столкнетесь с ним, то увидите

перед собой огромный ворох новых проблем. Тем не менее порой нет другого выбора: некоторые приложения просто требуют быть распределенными по следующим причинам:

- *данные*, используемые приложением, — распределены;
- *вычисление* — распределено;
- *пользователи* приложения — распределены.

Распределенные данные

Некоторые приложения должны выполняться на множестве компьютеров, потому что данные, к которым они должны обратиться, существуют также на множестве компьютеров. Владелец может разрешить дистанционный доступ к данным, но хранятся они как локальные. Таковы исторические (или иные) причины, определяющие распределенную организацию данных.

Распределенные вычисления

Некоторые приложения выполняются на множестве компьютеров, чтобы воспользоваться преимуществом параллельного вычисления для решения специфических задач, например, связанных с дешифровкой. Другие приложения также могут выполняться на множестве компьютеров, но чтобы использовать преимущества некоторых специфических систем для реализации различных частей своего алгоритма. Распределенные приложения всегда выигрывают при использовании неограниченной масштабируемости и неоднородности распределенной системы.

Распределенные пользователи

Некоторые приложения распределены по множеству компьютеров, потому что пользователи связываются и взаимодействуют друг с другом через это приложение. Каждый пользователь выполняет фрагмент распределенного приложения на его собственном или чужом компьютере и активно использует общедоступные объекты, которые обычно выполняются на одном или большем количестве серверов. Типичная архитектура для этого вида приложений иллюстрируется на рис. 3.2.

Как уже говорилось, CORBA является идеальной архитектурой для создания таких приложений. Актуальность ее в наши дни активно востребована развитием Интернета и систем электронной коммерции, где многие функции должны быть распределены по сети и где имеется в изобилии множество уже работающих объектных модулей, существующих в виде ActiveX- или JavaBean-компонентов.

Брокер объектных запросов, ORB, вручает запрос объекту и возвращает любой результат клиенту (рис. 3.3).

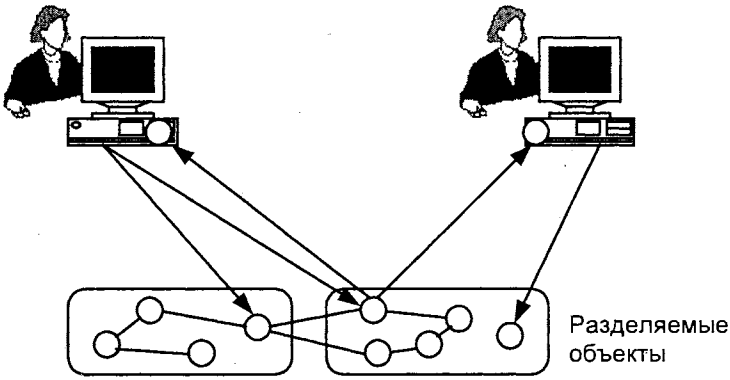


Рис. 3.2. Распределенные пользователи

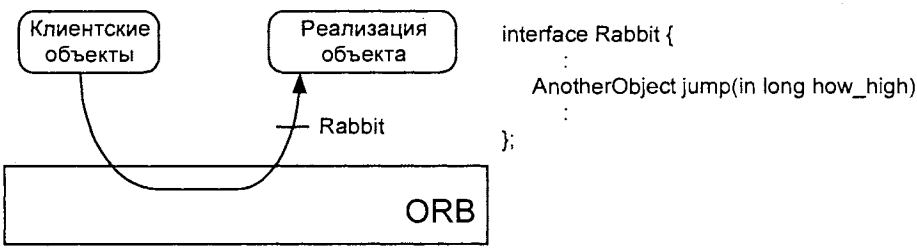


Рис. 3.3. Механизм взаимодействия: клиент запрашивает объект через посредника ORB

От клиента к серверу

Поскольку CORBA является стандартом для создания распределенных приложений, когда компьютеры через удаленный доступ вызывают программные методы и объекты друг друга, в ней четко прописан уровень межсетевых взаимодействий вне всякой зависимости от местонахождения, языка и архитектуры.

Можно четко сформулировать ключевые особенности CORBA-стандарта:

- CORBA-объекты могут находиться в любом месте сети;
- CORBA-объекты могут взаимодействовать с другими CORBA-объектами на любых платформах;
- CORBA-объекты могут быть написаны на любом языке программирования, для которого имеется интерфейс, реализуемый IDL-компилятором (таковые имеются для Java, C++, C, SmallTalk, COBOL и Ada).

Используя CORBA, вы можете создавать распределенные системы, где различные компоненты (интерфейсы пользователей, бизнес-алгоритмы, доступ

к базам данных и т. д.) будут паковаться в раздельных программах, выполняющихся на различных машинах. Каждый компонент будет связываться с другим только через объявленный интерфейс. При этом могут отлаживать и поддерживать части программ отдельно.

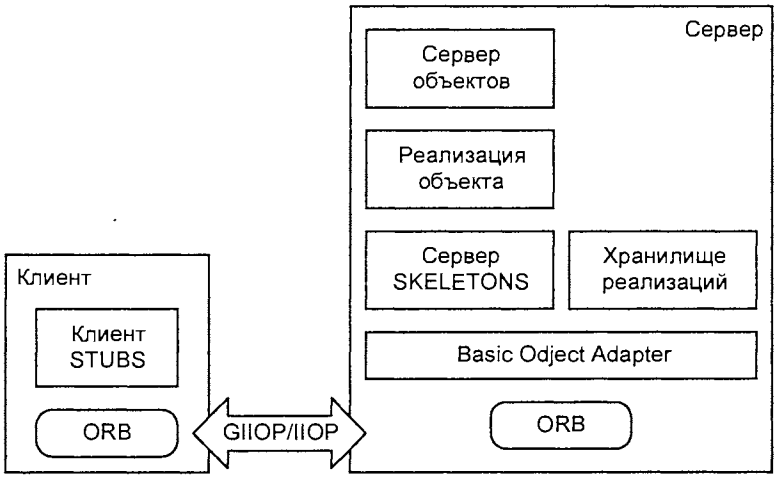


Рис. 3.4. Архитектура CORBA-совместимого приложения, вызывающего множество объектов с множества компьютеров

Клиентская часть CORBA

Когда CORBA-совместимый объект использует метод, находящийся в другом объекте, он вызывает стаб (stub), т. е. некую пустую оболочку требуемого объекта, его интерфейсную часть, находящуюся в специальном stub-файле. Этот стаб использует локальный ORB, посылает запрос и получает ответ. Объекту нет никакой необходимости знать подробности относительно ORB или действительного местонахождения сервера. Все CORBA-клиенты и серверы соединяются через брокеров ORB, используя их как посредников.

JDK 1.2 содержит ORB, который позволяет создавать CORBA-совместимые программы на Java.

Клиентский стаб создается при первом определении серверного интерфейса в клиенте при помощи языка определения интерфейсов IDL. CORBA IDL определяет тип объекта, его методы, свойства и т. д. CORBA IDL — это абсолютно нейтральный и чисто декларативный язык. В дальнейшем вы увидите, как легко создаются примеры на IDL.

Стаб затем использует локальный ORB, посылает запрос и получает ответ. ORB предоставляет своеобразную шину для прохождения обмена структурированными данными. Эта шина использует IIOP как транспортный протокол между брокерами ORB.

Серверная часть CORBA

При извлечении запроса IIOP сервер ORB использует так называемый *Basic Object Adapter* (BOA) в комбинации с *репозитарием выполнения* (Implementation Repository — набором конкретных классов и библиотек, имеющихся на сервере) для передачи параметров запроса к требуемому серверному объекту через серверный стаб. Иногда спрашивают — зачем так сложно? Ответ прост, чтобы легче было программировать независимые куски кода, сосредотачиваясь только на сути кода, а не на разрешении системных вопросов, связанных с сетевой спецификой и межпрограммном взаимодействии. О них пользователь вообще не должен задумываться, в точности так же, как он не задумывается о внутреннем устройстве телефонной трубки и телефонной станции, через которые проходит голосовой сигнал.

- **Basic Object Adapter**

  BOA — это набор системных сервисов ORB для установления соединения с серверными объектами по их идентификаторам (объектные ссылки).

- **Server IDL Stub**

  Служит для обеспечения интерфейса к каждому сервису. На серверной стороне такой стаб, чтобы отличить его от клиентского стаба, называется *СКЕЛЕТОН*.

- **Implementation Repository**

  Это набор классов на стороне сервера, доступный всем распределенным клиентам через систему объектных ссылок.

Сервисы CORBA

Важной частью стандарта CORBA является определение набора распределенных услуг (сервисы CORBA), чтобы поддерживать интеграцию и взаимодействие распределенных объектов. Они определены как стандартные объекты CORBA с интерфейсами IDL и иногда еще упоминаются как *объектные сервисы* (рис. 3.5).

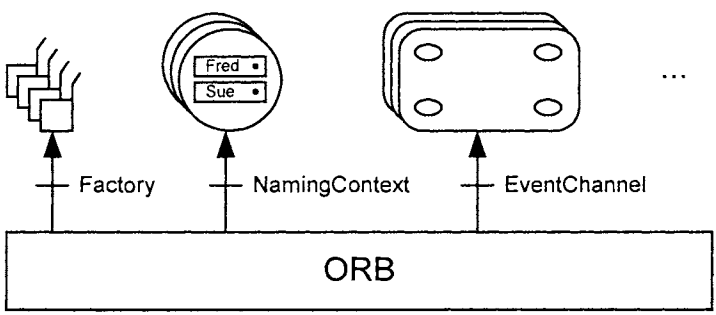


Рис. 3.5. Объектные сервисы

Имеется множество CORBA-сервисов. Самые популярные из них приведены в табл. 3.1.

Таблица 3.1. *Наиболее популярные сервисы CORBA*

Сервис	Описание
Жизненный цикл объекта (Object life cycle)	Определяет, каким образом CORBA-объекты будут созданы, удалены, перемещены или скопированы
Именование (Naming)	Определяет способ символического обозначения CORBA-объектов
События (Events)	Обработка событий
Отношения (Relationships)	Определяют отношения между объектами (главный, подчиненный, связной и т. д.)
Преобразование объектов из одной формы в другую (Externalization)	Координирует преобразование объектов из внешних форм во внутренние и наоборот
Транзакции (Transactions)	Координируют доступ к CORBA-объектам
Контроль доступа (Concurrency Control)	Обеспечивает обслуживание блокировки объектов CORBA
Свойство (Property)	Поддерживает ассоциацию пары: имя-значение для объектов CORBA
Продавец (Trader)	Поддерживает поиск CORBA-объектов, основанный на афишируемых свойствах объекта
Запрос (Query)	Поддержка запросов к объектам

CORBA-продукты

Не забывайте, что CORBA — это всего лишь спецификация. Что касается программных изделий, реализующих архитектуру CORBA, то их имеется множество. Отдельные продавцы обеспечивают свои брокеры CORBA и IDL для отображения в различные языки программирования. Все они обязаны поддерживать Java. В табл. 3.2 приведены наиболее известные реализации CORBA.

Таблица 3.2. *Наиболее известные реализации CORBA*

ORB	Описание
Java 2 ORB	Java 2 ORB поставляется в составе Sun's Java 2 SDK. Там отсутствуют некоторые особенности полной спецификации

Таблица 3.2 *(окончание)*

ORB	Описание
VisiBroker for Java	Популярный Java ORB из Inprise Corporation. VisiBroker встроен во многие другие продукты. К примеру, именно он был встроен в браузер Netscape Communicator
OrbixWeb	Добротный Java ORB из Iona Technologies (для Unix)
WebSphere	Мощный сервер приложений со встроенным ORB от IBM
Free or shareware ORBs	CORBA-реализации для различных языков программирования доступны для загрузки по сети Интернет из различных источников

Объектные Web-модели

Результатом последнего десятилетия в мире, связанном с проектированием программных архитектур, было появление новой Web-объектной модели. На рис. 3.6 показана наиболее типичная современная Web-объектная модель, которая, в отличие от традиционных клиент-серверных приложений, целиком погружена в среду Интернет/интранет. Как и всегда, первый уровень целиком принадлежит клиенту. Но теперь типов клиента может быть два:

- традиционный браузер-ориентированный клиент;
- чистый Java-клиент, которого принято теперь ассоциировать с будущим универсальным и мобильным клиентом.

На втором уровне действует практически любой Web-сервер, который автоматически получает статус сервера приложений, если он способен поддерживать клиентов HTTP и CORBA, предоставляя им различные сервисы для взаимодействия с третьим уровнем.

Третий уровень представлен в виде традиционных серверов.

Средний уровень CORBA/HTTP поддерживается почти на всех серверных платформах, включая семейство Unix, Windows NT, OS/2, NetWare, MacOS, OS/400, MVS (мэйнфрейм) и Tandem NonStop Kernel. CORBA-объекты действуют при этом как серверы приложений; они инкапсулирут бизнес-логику. Взаимодействие с клиентами осуществляется через Java ORB, который, чтобы использовать в качестве среды передачи Интернет, должен быть ориентирован на протокол IIOP. Разумеется, этим не ограничивается использование CORBA-объектов. Они могут взаимодействовать с существующими серверами приложений и серверами БД, посредством SQL и использования других форм *middleware* (программного обеспечения среднего звена).

Средний уровень должен использовать координаторы или мониторы транзакций. В них также не имеется недостатка, достаточно назвать: IBM's Busi-

ness Object Server Solution (BOSS), BEA Systems' Tuxedobased ObjectWare или MTS, JTS и т. д. и т. п. Что такое сервер-ориентированные компоненты? Ими могут быть серверные объекты CORBA, реализующие минимальный набор компонентного сервиса. Лучшим примером являются картриджы Oracle, которые именуются CORBA-объектами, потому что они транзакционны, безопасны и способны к обработке событий.

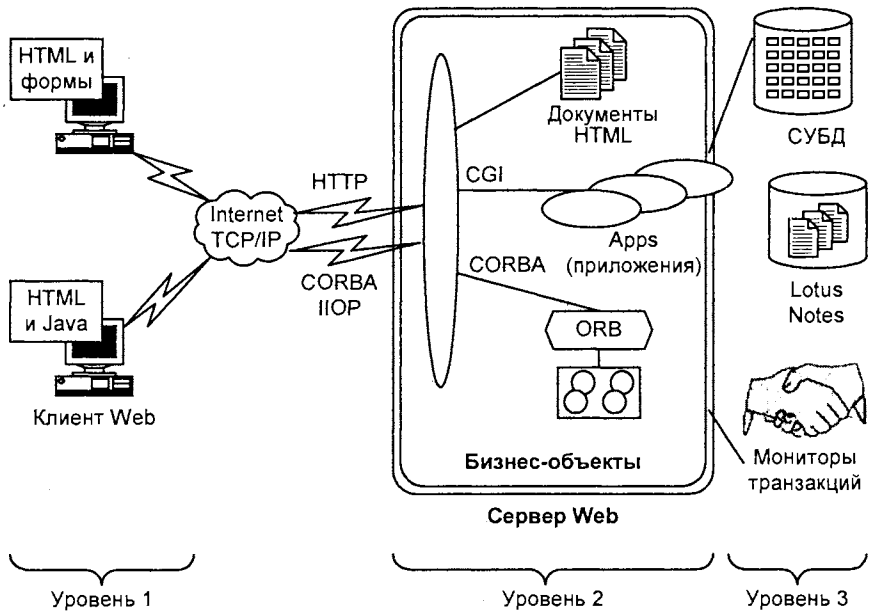


Рис. 3.6. Объектная Web-модель, основанная на Java-клиенте и CORBA ORB

Компоненты на сервере должны обладать способностью к динамическому построению. Это означает, что необходимо обеспечить динамический интерфейс, который будет допускать использование визуальных инструментальных средств. Например, объект может обеспечить пиктограмму, являющуюся некоторым средством управления. Когда вы выбираете пиктограмму с помощью мыши, объект должен отобразить методы, которые он поддерживает, и CORBA-события, которые он способен обработать. Все это должно выполняться через самоанализ (интроспекцию, — т. е. компонент может сообщать вам кое-что относительно себя).

Третий уровень — это уровень общения CORBA-объектов с серверами БД или системами групповой работы с документами типа Lotus Notes. На этом уровне бизнес-объекты CORBA должны полностью вытеснить некогда весьма популярные CGI-приложения.

Java-клиент может непосредственно взаимодействовать с CORBA-объектом путем использования IIOP ORB. Это еще один довод в пользу более естест-

венных объектных коммуникаций, против громоздкого и неудобного HTTP/CGI на среднем уровне. Подобно HTTP, CORBA IIOP использует Интернет в качестве позвоночника (backbone). Это означает, что и IIOP и HTTP прекрасно уживаются в Сети. HTTP может использоваться для загрузки Web-страниц, апплетов и картинок; CORBA должна использоваться для клиент-серверных коммуникаций Java.

Взаимодействие типа клиент-сервер в объектной модели Web

На рис. 3.7 показано, как типичный Web-клиент взаимодействует с объектно-ориентированным сервером. Первым делом, браузер загружает полученную с сервера HTML-страницу. Страница может включать Java-апплеты, которые также загружаются. HTTP-сервер загружает апплеты в виде байт-кодов. Апплет размещается в памяти и исполняется при помощи виртуальной Java-машины. Апплет может вызывать серверные CORBA-объекты. Java-апплет может включать в себя сгенерированные при помощи IDL стабы и вызывать нужные ему объекты с сервера посредством обращения к брокеру объектных запросов ORB или, в качестве альтернативы, он может использовать интерфейс CORBA Dynamic Invocation Interface, чтобы сгенерировать запросы к

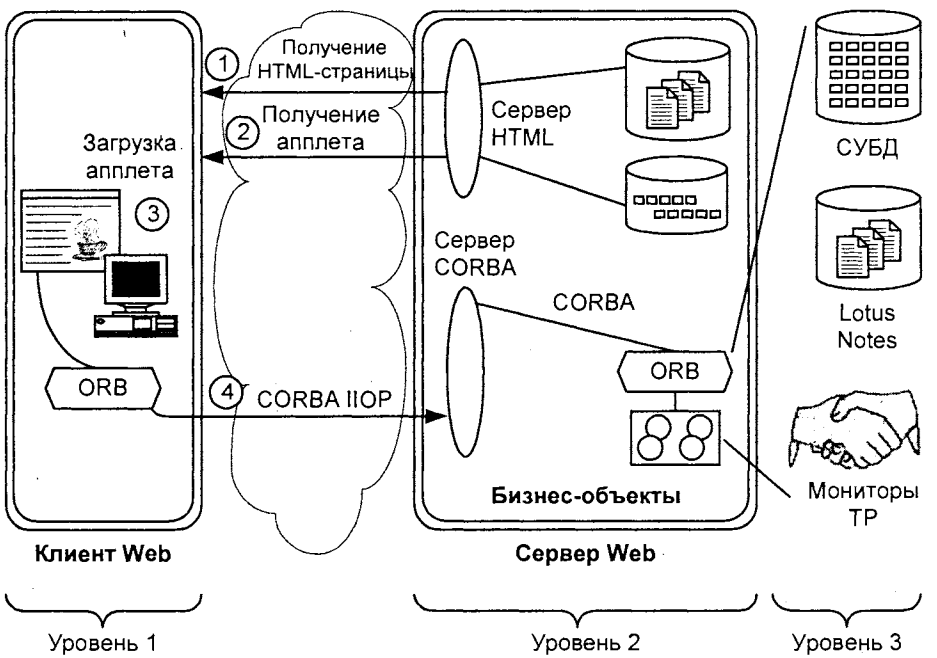


Рис. 3.7. Как HTTP, Java и CORBA работают вместе

серверу "на лету". Сеанс между небольшим приложением Java и CORBA-объектами будет сохраняться, пока любая сторона не решит оборвать существующее соединение.

Имеются ли другие игроки на этом рынке

В 4-й части книги "Client/Server Programming with Java and CORBA" (John Wiley & Sons, 1997) приводится детальное описание пяти основных инфраструктур, таких как:

- Streamed Sockets
- HTTP/CGI
- Java RMI (Remote Method Invocation)
- CORBA/IIOP
- DCOM/ActiveX

Авторы книги создали клиент-серверное приложение для использования во всех вышеперечисленных технологиях и свели результаты в таблицу (рис. 3.8).

Судя по этой таблице CORBA — вне конкуренции, вернее у нее только один серьезный соперник — DCOM.

В сценарии с самым плохим исходом DCOM мог бы победить, если бы группа компаний, входящая в сетевую объектную коалицию, развалилась.

Но, как ни странно, именно CORBA становится для этих компаний консолидирующим элементом и мощной архитектурой, которая связывает вместе программы и продукты, формирующие объектную Сеть. Например, ORB Netscape прекрасно взаимодействует с картриджами Oracle, так же, как могут работать совместно мониторы транзакций IBM, BEA System и Hitach, потому что они используют CORBA-архитектуру.

В параллельном мире Microsoft формирует собственный вариант объектной Сети, основанной на DCOM и ActiveX. Сервер транзакций Microsoft — MTS (его прежнее кодовое название было Viper, что означает "гадюка") — это секретное оружие Microsoft, выкованное ею для победы. В настоящее время Microsoft напоминает гигантский корабль с прилепившимися к нему тысячами мелких ботиков, бросивших якорь в его огромной тени.

В противоположном лагере — группа компаний: Netscape, Oracle, JavaSoft и IBM/LOTUS. Этот лагерь также заполняется сотнями продавцов программного обеспечения, которые обеспечивают инструментальные средства, компоненты и услуги. Нужно накопить достаточную критическую массу, чтобы привлечь покупателей с независимыми от производителей целями и потребностями.

- Oracle продает ПО на 4 миллиарда долларов ежегодно и полностью принял CORBA в качестве платформы для Архитектуры Сетевых Вычисле-

ний. Вся линейка программного обеспечения Oracle, от машины базы данных до сохраненных процедур, инструментальных средств и Интернета формируется на базе объектов CORBA. Например, машина базы данных будет переведена в компоненты, использующие CORBA. Третьи фирмы будут способны расширять базу данных, используя CORBA-компоненты по имени картриджей. Весь арсенал Oracle, включая Developer 2000 и Sedona, разработан под CORBA для клиента и сервера. Наконец, Oracle создает свои собственные сервисы ORB и CORBA/IIOP, которые включены в Oracle Web Server.

Сравнительный анализ и оценка CORBA и JAVA ORBS					
Характеристика	**CORBA/IIOP**	**OCOM**	**RMI**	**HTTP/CGI**	**SOCKETS**
Уровень абстракции	★★★★	★★★★	★★★★	★★	★
Прозрачная интеграция Java	★★★★	★★★	★★★★	★★	★★
Поддержка OS	★★★★	★★	★★★★	★★★★	★★★★
Реализация Java	★★★★	★	★★★★	★★★★	★★★★
Поддержка типизированных параметров	★★★★	★★★★	★★★★	★	★
Простота конфигурирования	★★★	★	★★★	★★★	★★★
Вызов распределенных методов	★★★★	★★★	★★★	☆	☆
Статус вызовов	★★★★	★★★	★★★	☆	★★
Динамическое распознавание и поддержка метаданных	★★★★	★★★	☆	☆	☆
Динамические вызовы	★★★★	★★★★	★	☆	☆
Производительность (удаленный доступ Pings)	★★★★ 3.3 мсек	★★★ 3.9 мсек	★★★ 5.5 мсек	☆ 603.8 мсек	★★★★ 2.0 мсек
Безопасность	★★★★	★★★★	★★★	★★★	★★★
Транзакции	★★★★	★★★	☆	☆	☆
Неизменяемые ссылки на объект	★★★★	★	☆	☆	☆
Именование на базе URL	★★★	★	★★	★★★★	★★★

Рис. 3.8. Таблица с результатами использования пяти основных инфраструктур

- JavaSoft формирует Enterprise Java с использованием CORBA/IIOP в качестве основы. SUN ORB уже включен в состав JDK.
- IBM/LOTUS формирует сеть с кроссплатформенной вычислительной инфраструктурой на базе CORBA/JAVA. IBM Visual Age является средством разработки CORBA/JAVA для клиентов и серверов на любых плат-

формах. Наконец, IBM разрабатывает масштабируемый координатор транзакций BOSS для управления объектами CORBA/JAVA и создания средств middleware.

По сути, битва за объектную сеть только еще начинается. Кто победит в этой ожесточенной конкурентной борьбе может показать только время.

Примеры с использованием CORBA

Чтобы лучше понять механизм действия CORBA, напишем небольшое CORBA-приложение по рекомендациям и примерам из Magelang Institute, приводимых на сайте **developer.java.sun.com**. В качестве приложения возьмем наиболее типичный случай использования новейших технологий в сфере электронного бизнеса — торговлю акциями на бирже. Данный пример интересен тем, что даже в небольшом фрагменте этого сложного приложения мы будем применять методы, которые позволят организовать работу нескольких тысяч одновременно работающих пользователей. В чем специфика распределенных Интернет-приложений? В том, что к ним могут подключаться и использовать их с самых разнообразных мест десятки тысяч клиентов. Особенно ярко это проявляется в торговле акциями.

Приложение *Акции*

В этом распределенном приложении будут продемонстрированы возможности Java и CORBA. По сути это только каркас приложения, которое можно совершенствовать и расширять бесконечно.

Данное приложение позволяет множеству пользователей наблюдать изменения текущего курса акций ведущих компаний. Пользователь обеспечен списком доступных котировок, идентифицированных символами компаний. Он может выбрать акцию и затем нажать кнопку **View stock** (Просмотр) (рис. 3.9).

При нажатии на кнопку **View stock** (Просмотр) пользователь может посмотреть отчет о текущей цене акции, последнем изменении цены, объем продаж и т. д. Отчет выводится в графическом виде и автоматически модифицируется (рис. 3.10).

Пользователь имеет возможность установить сигнал тревоги, когда цена отслеживаемой им акции падает ниже допустимой отметки или, наоборот, превышает некоторый порог цены (рис. 3.11).

В этом случае приложение известит пользователя о наступлении такого события.

В принципе приложение **Акции** должно иметь следующие объекты:

`Stock` Распределенный объект, содержащий информацию о соответствующих акциях

`StockPresentation`	Распределенный объект, в графической форме выдающий информацию об акциях для клиента
`Alarm`	Распределенный объект, устанавливающий пороговые пределы для конкретного клиента и выбранной им акции
`AlarmPresentation`	Распределенный объект, извещающий пользователя о наступлении некоторого события

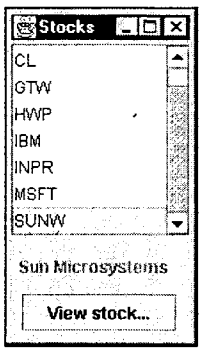


Рис. 3.9. Окно для просмотра акций компаний

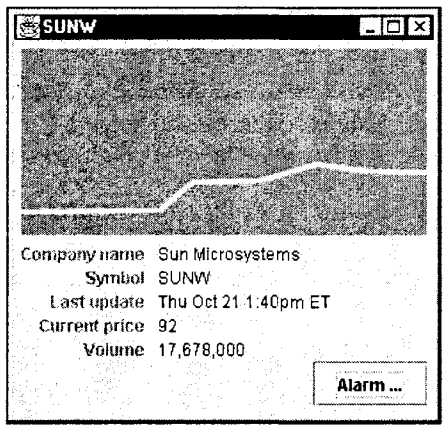


Рис. 3.10. Отчет по акциям SUNW

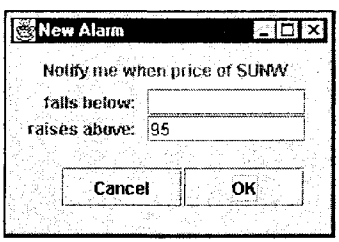


Рис. 3.11. Окно ввода пределов цены акций

Рассмотрим подробно объект `Stock` в качестве распределенной модели.

Реализация CORBA-клиента

Опишем IDL-интерфейсы для объекта `Stock`.

```
module StockObjects {

  struct Quote {
    string symbol;
    long at_time;
    double price;
```

```
    long volume;
  };

  exception Unknown{};

  interface Stock {

    // Возвращает информацию об акции.
    Quote get_quote() raises(Unknown);

    // Устанавливает текущую котировку.
    void set_quote(in Quote stock_quote);

    // Обеспечивает некоторую информацию,
    // например, имя компании.
    readonly attribute string description;
  };

  interface StockFactory {

    Stock create_stock(
      in string symbol,
      in string description
   );
  };
};
```

В вышеприведенном примере мы имеем IDL-модуль, который содержит:

- структуру `Quote`;
- исключение `Unknown`;
- интерфейс `Stock`;
- интерфейс `StockFactory`.

Отметим, что параметры в определении методов маркируются ключевыми словами `in`, `out` или `inout`. Ключевое слово `in` указывает, что данные передаются от клиента к объекту. Ключевое слово `out` обозначает то, что данные возвращаются от объекта к клиенту. Ключевое слово `inout` указывает, что данные могут передаваться от клиента к объекту и затем возвращаться назад к клиенту. IDL-декларации компилируются с помощью IDL-компилятора и отображаются в конкретный язык программирования, для которого создан IDL-компилятор. Мы будем использовать IDL из состава JDK1.2 SDK, который, естественно, отображается в Java.

Объектные ссылки и запросы

Клиенты выдают запросы к CORBA-объектам, используя объектную ссылку. Объектная ссылка идентифицирует распределенный объект, который полу-

чит запрос. Ниже приводится фрагмент кода на языке Java, в котором применяется ссылка на объект `Stock`, и затем она используется, чтобы получить информацию об акции. Обратите внимание, что в данном фрагменте кода непосредственно не применяются типы CORBA, взамен этого есть Java-типы, которые были произведены компилятором IDL для Java-компилятора.

```
Stock theStock = ...
try {
    Quote current_quote =
              theStock.get_quote();
} catch (Throwable e) {
}
```

Объектные ссылки могут передаваться по распределенной объектной системе как параметры операций и возвращаться как результаты запросов. Например, обратите внимание, что интерфейс `StockFactory` определяет операцию `create ()`, которая возвращает экземпляр объекта `Stock`. Ниже приводится фрагмент кода на Java, который выдает запрос на конкретный объект по имени фирмы и получает возникающую в результате ссылку на объект `Stock` (т. е. информацию по стоимости акции этой фирмы).

```
StockFactory factory = ...
Stock theStock = ...
try {
   theStock = factory.create(
                "GII",
                "Global Industries Inc.");
} catch (Throwable e) {
}
```

Обратите внимание, что выдача запроса к объекту CORBA — это вовсе не то, что выдача запроса к объекту Java на локальной машине. Основное отличие то, что объекты CORBA могут быть где угодно! CORBA-система обеспечивает прозрачность местоположения, которая подразумевает, что клиент может не сообщать в запросе — находится ли его объект в том же самом процессе, на той же самой машине или где-нибудь на другом конце планеты.

Другое отличие от локального объекта Java то, что время существования (life time) объекта CORBA не связано ни с процессом, который клиент выполняет, ни с процессом, в котором объект CORBA выполняется. Объектные ссылки сохраняются; они могут быть сохранены в виде строки и вновь создан из строки.

Следующий фрагмент преобразует объектную ссылку `Stock` в строку:

```
String stockString =
     orb.object_to_string(theStock);
```

Строка может быть сохранена или передана в виде сообщения другому объекту. Любой клиент может преобразовывать строку обратно в объектную ссылку и выдавать запрос к полученному распределенному объекту.

Нижеследующий фрагмент преобразовывает строку обратно в объектную ссылку объекта `Stock`:

```
org.omg.CORBA.Object obj =
    orb.string_to_object(stockString);
Stock theStock = StockHelper.narrow(obj);
```

Обратите внимание, что возникающий в результате тип метода `string_to_object ()` — это `Object`, а не `Stock`. Вторая строка сужает тип объектной ссылки из `Object` в `Stock`. IDL поддерживает иерархию интерфейсов; вызов метода `narrow ()` — это операция в иерархии.

Возьмемся, наконец, за дело и напишем очень облегченное CORBA-приложение, реализующее выдачу информации об акциях компаний. Оно будет состоять из 4-х программ:

- **stock.idl** — описание интерфейсов на IDL;
- **StockClient.java** — программа клиент;
- **StockServer.java** — программа сервер;
- **StockImpl.java** — реализация интерфейсов.

1. Программа stock.idl

```
module StockObjects {
       struct Quote {
string symbol;
       // Обозначение данного пакета.
              long at_time;
       // Время обновления цены.
              double price;
       // Текущая цена пакета акций.
              long volume;
              // Количество акций.
       };
    exception Unknown{};
    interface Stock {
// Текущее обозначение пакета.
              Quote get_quote() raises(Unknown);
              void set_quote(in Quote stock_quote);
              // Описание (например, имя компании).
              readonly attribute string description;
};
};     // конец модуля StockObjects
```

2. Программа StockClient.java

```
import java.io.*;
import org.omg.CORBA.*;
import StockObjects.*;

public class StockClient {
    public static void main(String args[]) {
        if (args.length!=1) {
            System.err.println("Required argument: stock ior file");
            return;
        }
        try{
                // Создать и инициализировать ORB
                ORB orb = ORB.init(args, null);

                // Считать ссылку на объект stock, преобразованную в строку
            // Подразумевается, что создан сервером
                BufferedReader in = new BufferedReader(new
                                                FileReader(args[0]));
                String ior = in.readLine();
                in.close();

                // Обратное преобразование объекта
                org.omg.CORBA.Object obj = orb.string_to_object(ior);
            // Сужаем до объекта stock
            Stock theStock = StockHelper.narrow(obj);

    // Вызов объекта stock и получение описания и текущей цены
            Quote currentQuote = theStock.get_quote();
            System.out.print(theStock.description());
            System.out.println(" is currently selling at
"+currentQuote.price);
            System.out.println("Volume is "+currentQuote.volume);
           } catch (Exception e) {
                e.printStackTrace(System.out);
           }
    }
}
```

3. Программа StockServer.java

```
import java.io.*;
import StockObjects.*;

// Функция main для сервера.

public class StockServer {
```

```
    public static void main(String[] args) {
        if (args.length!=1) {
            System.err.println("Required argument: stock ior file");
            return;
        }
        try {
            // Инициализировать ORB.
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

            // Создать объект stock.
            StockImpl theStock = new StockImpl("GII","Global Industries
Incorporated");
                        orb.connect(theStock);

            // Выдать цену.
            Quote quote = new Quote();
            quote.price=28.0;
            quote.symbol="GII";
            quote.volume=0;
            quote.at_time=System.currentTimeMillis();
            theStock.set_quote(quote);

            // Напечатать ссылку на объект в виде строки
            System.out.println( "Created GII:\n" +
orb.object_to_string(theStock));

            // Записать объект, преобразованный в строку, на диск
            PrintWriter out = new PrintWriter(new BufferedWriter
                                        (new FileWriter(args[0])));
            out.println( orb.object_to_string(theStock));
            out.close();

                        // Ожидание
                        java.lang.Object sync = new java.lang.Object();
                        synchronized (sync) {
                            sync.wait();
            }

        } catch (Exception e) {
                System.err.println("Stock server error: " + e);
                e.printStackTrace(System.out);
            }
    }
}
```

4. Программа StockImpl.java

```
import StockObjects.*;
public class StockImpl extends StockObjects._StockImplBase {
    private Quote _quote=null;
    private String _description=null;
    public StockImpl(String name,String description) {
        super();
        _description = description;
    }

    public Quote get_quote() throws Unknown {
        if (_quote==null) throw new Unknown();
        return _quote;
    }
// *** ваш метод, реализующий IDL-операцию
//     для установки квоты
     public void set_quote(Quote quote) {
        _quote = quote;
    }
    public String description() {
        return _description;
    }

}
```

Чтобы скомпилировать эти программы, воспользуемся стандартным Java и компилятором Sun JavaIDL:

```
idltojava -fno-cpp stock.idl
javac-classpath . *.java
```

В результате выполнения этих команд у вас будет создан каталог StockObject в каталоге, где лежат файлы примера, и в него компилятор IDL поместит сгенерированные на Java программы, реализующие описанные интерфейсы с использованием CORBA. Таких программ будет несколько:

```
Directory of D:\My_book\Examples\JavaBeans\StockObjects

09.04.00  14:08         <DIR>          .
09.04.00  14:08         <DIR>          ..
09.04.00  14:08                      0 aaa.txt
09.04.00  14:03                    603 Quote.java
09.04.00  14:03                   2589  QuoteHelper.java
09.04.00  14:03                    841 QuoteHolder.java
09.04.00  14:03                    432 Stock.java
09.04.00  14:03                   1873  StockHelper.java
```

```
09.04.00  14:03                841 StockHolder.java
09.04.00  14:03                358 Unknown.java
09.04.00  14:03               1605 UnknownHelper.java
09.04.00  14:03                859 UnknownHolder.java
09.04.00  14:03               3232 _StockImplBase.java
09.04.00  14:03               2309 _StockStub.java
```

В принципе, вам вовсе необязательно во всех подробностях вникать в эти сгенерированные программы. Они вспомогательные. Вместо них мы будем использовать более компактную программу StockImpl.java, которая является расширением сгенерированного класса `_StockImplBase` и реализует все описанные интерфейсы.

Чтобы выполнить эти программы, запустите сначала сервер из командной строки:

```
java -cp . StockServer myfile.txt
```

А потом клиента командой:

```
java -cp . StockClient myfile.txt
```

Имя файла `myfile.txt` в качестве аргумента требуется для того, чтобы сообщить серверу имя файла, в котором он должен сохранять объектную ссылку в виде текстовой строки. Именно по этой объектной ссылке, в нашем случае передаваемой через файл, клиент и будет находить требуемые ему объекты (рис. 3.12).

Что же происходит в процессе взаимодействия сервера и клиента?

IDL-компилятор генерирует клиентскую часть программы, так называемые *клиент-стабы* (stubs), которые представляют собой объекты CORBA на локальной машине. Сгенерированный код также представляет в языке программирования Java все интерфейсы IDL и типы данных, используемые для выдачи запросов к удаленным объектам. Самостоятельный клиентский код, таким образом, зависит от сгенерированного на основе IDL кода Java (рис. 3.13).

Для серверной части программы, точно таким же способом, как и для клиента, также на основе IDL генерируется Java-код в виде так называемых скелетонов (рис. 3.14).

Именно через стабы и скелетоны будет прозрачно осуществляться удаленное межпрограммное взаимодействие. Тут возникает вопрос — а зачем в нашем случае потребовался еще один класс `StockImpl`, реализованный при помощи программы StockImpl.java? Дело в том, что объектные реализации CORBA полностью невидимы для их клиентуры. Клиент может зависеть только от интерфейса IDL. Однако в языках программирования Java и C++ дело обстоит не совсем так. Имеются различные реализации CORBA от разных

фирм производителей и различные IDL-компиляторы, отображающие стандарт CORBA в разнообразные языки программирования. Введение объекта `implementor` предоставляет гораздо большую свободу. Объектные реализации можно обеспечивать в ряде различных языков программирования. Тот же самый интерфейс может быть реализован многими способами. Не имеется никакого ограничения. Например, можно предложить следующие возможные реализации интерфейса `Stock` в приложении **Акции**:

- написать его на Java, чтобы он получал значения из некоторого коммерческого источника;
- написать его на C++, чтобы он обращался к базе данных через Интернет;
- написать его на SmallTalk.

```
Cmd.exe
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\WINNT\system32>d:

D:\>cd mybook

D:\MYBOOK>cd examples

D:\MYBOOK\examples>cd corba

D:\MYBOOK\examples\Corba>java -cp . StockClient myfile.txt
Global Industries Incorporated is currently selling at 28.0
Volume is 0

D:\MYBOOK\examples\Corba>
```

```
Cmd.exe - java -cp . StockServer myfile.txt
idltojava: unknown flag: -fn0-cpp
idltojava: aborted due to earlier errors

D:\MYBOOK\examples\Corba>idltojava -fno-cpp stock.idl

D:\MYBOOK\examples\Corba>javac -classpath . *.java
Note: .\StockObjects\_StockImplBase.java uses or override
compile with "-deprecation" for details.
1 warning

D:\MYBOOK\examples\Corba>java -cp . StockServer myfile.tx
Created GII:
IOR:000000000000001b49444c3a53746f636b4f626a656374732f537
0001000000000000002c0001000000000000006776f726b3500051400000
0e380000000080000000000000000
```

Рис. 3.12. Выполнение примера

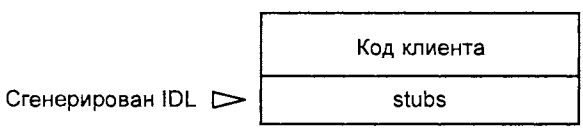


Рис. 3.13. Схема генерации клиентской части программы

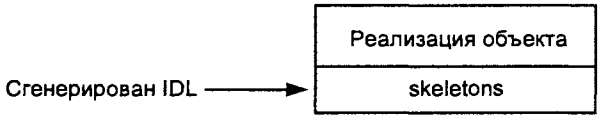


Рис. 3.14. Схема генерации северной части программы

В дополнение к файлам, созданных для клиента, IDL также генерирует скелетный класс для объектной реализации. Скелетон — это точка входа в распределенный объект. Благодаря ему производится маршаллизация данных, т. е. выполнение запрашиваемого метода и возвращение результатов выполнения клиенту, будто он обратился к обычной процедуре на своей машине. Чтобы выполнить объект CORBA на Java, разработчик просто реализует дополнительный класс Java, который расширяет сгенерированный скелетный класс и обеспечивает метод для каждой операции в интерфейсе. В нашем примере, IDL-компилятор генерирует скелетный класс `_StockImplBase`, а мы обращаемся к нему опосредованно — через класс `StockImpl`.

Разумеется, взятое нами в качестве примера CORBA-приложение еще далеко до совершенства. Объектная ссылка от клиента к серверу передается через файл. Это далеко не самый лучший способ хранения и распространения объектных ссылок. В упомянутых нами CORBA-сервисах, например, имеется такой замечательный сервис, как Naming Service, о котором можно подробно прочитать в документации по JDK1.2. В качестве примера использования этого сервиса приведем еще одно CORBA-приложение — знаменитую программу Hello World!, с которой начинается изучение любого языка программирования. В этом примере и в его усложненном варианте с использованием `CallBack` применяется сервис Naming Services для передачи объектных ссылок.

Программа Hello World!

Эта программа демонстрирует основные приемы для построения распределенного приложения CORBA на примере классического для всех языков программирования образца — программы Hello World!, используя Java IDL. Будет построено как Java-приложение, так и Java-апплет. Программа Hello World! имеет единственную функцию, возвращающую строку, которая должна быть напечатана. Схема работы проиллюстрирована на рис. 3.15.

Основные операции распределенного приложения:

1. Клиент (апплет или приложение) вызывает метод `sayHello()` программы HelloServer.
2. ORB передает вызов к обслуживающему объекту, который зарегистрирован с IDL-интерфейсом.
3. Обслуживающий метод `sayHello()` выполняется и возвращает Java-строку.
4. ORB передает строку по сети к клиенту.
5. Клиент печатает полученную строку.

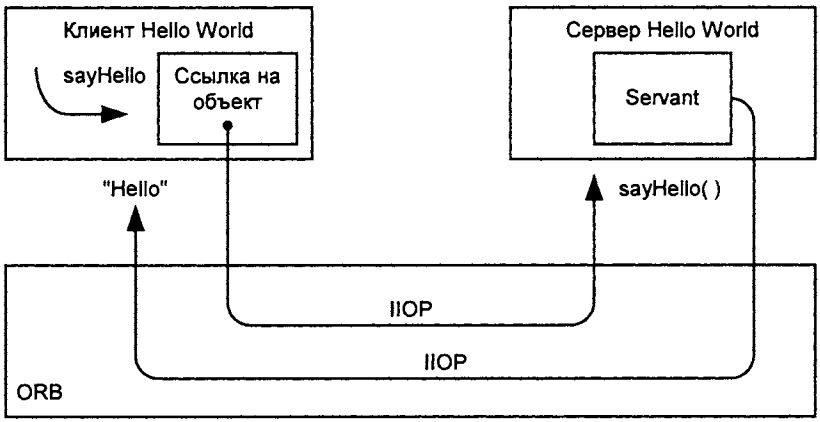


Рис. 3.15. Схема работы программы Hello World!

Определение интерфейса (Hello.idl)

Используется стандартный компилятор Sun IDL **idltojava**, который можно бесплатно скачать с узла **java.sun.com**.

Написание Hello.idl

Необходимо выполнить 3 шага:

1. Объявить CORBA IDL-модуль
2. Объявить интерфейс.
3. Объявить функции.

Объявление модуля

CORBA-модуль — это просто область имени.

```
module HelloApp
{
    // Здесь можете добавлять строки исходного кода.
};
```

Объявление интерфейса

Подобно интерфейсу Java, CORBA-интерфейсы декларируют API объектов, имеющих связь с другими объектами.

```
{
  interface Hello  // Добавьте
  {                // эти
                   // четыре
  };               // строчки.
};
```

Объявление функций

CORBA-функции — это некоторые действия, которые серверы выполняют от имени клиентов, вызывающих их. Каждый оператор функции в IDL генерирует соответствующий метод в сгенерированном Java-интерфейсе.

```
module HelloApp
{
  interface Hello
  {
    string sayHello();  // Добавьте эту строку.
  };
};
```

Отображение Hello.idl из IDL в Java

Выполните по командной строке:

```
idltojava Hello.idl
```

Результатом будет являться 5 файлов, среди которых:

- `Hello.java` — сгенерированная Java-программа:

  ```
  /* Hello.java as generated by idltojava */

  package HelloApp;
  public interface Hello
      extends org.omg.CORBA.Object {
      String sayHello();
  ```

- `_HelloImplBase.java`

  Абстрактный класс, так называемый скелетон на стороне сервера `server skeleton`. Серверный класс `HelloServant` расширяет класс `_HelloImplBase`.

- `_HelloStub.java`

  Класс на стороне клиента `client stub`, так называемый стаб (суррогат, заглушка).

- `HelloHelper.java`

  Этот заключительный класс обеспечивает дополнительные функциональные возможности, чтобы привести ссылки объекта CORBA к их соответствующим типам.

- `HelloHolder.java`

  Этот заключительный класс обеспечивает образец класса типа `Hello`. Он обеспечивает также функции для определения входных и выходных аргументов, которые CORBA может иметь, но которые не отображает в Java-семантике.

Разработка программы для сервера (HelloServer.java)

Программа для сервера образует два класса: обслуживание (servant) и сервер. Обслуживание реализует интерфейс `Hello IDL`. Каждый входящий запрос на обслуживание со стороны многочисленных клиентов обслуживается при помощи `HelloServant`. Этот класс является подклассом `_HelloImplBase`, который генерируется компилятором idltojava из описанного интерфейса IDL. `HelloServant` содержит один метод для каждой IDL-функции (метода) в нашем примере — см. метод `sayHello()`. Серверный класс имеет метод `main()`, который:

- создает ORB-образец;
- создает обслуживание (servant — объект CORBA `Hello`) и оповещает ORB об этом;
- получает CORBA-ссылку на имя объекта, для которого регистрируется новый CORBA-объект;
- регистрирует новый объект под именем `Hello`;
- ожидает вызовов для создания новых объектов.

```
//
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

class HelloServant extends _HelloImplBase
{
    public String sayHello()
    {
       return "\nHello world !!\n";
    }
}

public class HelloServer {

    public static void main(String args[])
    {
       try{
           // Создать и инициализировать ORB
           ORB orb = ORB.init(args, null);

           // Создать servant-объект и зарегистрировать его в ORB
           HelloServant helloRef = new HelloServant();
           orb.connect(helloRef);

           // Получить корневой контекст системы именования
           org.omg.CORBA.Object objRef =
```

```
            orb.resolve_initial_references("NameService");
        NamingContext ncRef = NamingContextHelper.narrow(objRef);

        // Связать ссылку на объект с именованием
        NameComponent nc = new NameComponent("Hello", "");
        NameComponent path[] = {nc};
        ncRef.rebind(path, helloRef);

        // Ожидание
         java.lang.Object sync = new java.lang.Object();
         synchronized (sync) {
             sync.wait();
         }

    } catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
    }
  }
}
```

Разработка для программы клиента (HelloClient.java)

Программа клиент выполняет следующее:

- создает ORB (помните, ORB — это брокер объектных запросов);
- получает ссылку по имени объекта, т. е. система адресует клиента к требуемому ему объекту;
- вызывает метод объекта `sayHello()` и печатает результат.

```
//

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class HelloClient
{
    public static void main(String args[])
    {
       try{
           // Создать и инициализировать ORB
           ORB orb = ORB.init(args, null);

            // Получить корневой контекст системы именования
            org.omg.CORBA.Object objRef =
              orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

```
            // Разрешить ссылку на объект в имя
            NameComponent nc = new NameComponent("Hello", "");
            NameComponent path[] = {nc};
            Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

            // Вызвать сервер Hello и напечатать результат
            String hello = helloRef.sayHello();
            System.out.println(hello);

        } catch (Exception e) {
            System.out.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

Разработка клиентской программы в виде апплета (HelloApplet.java)

Следущий пример — это просто реализация на Java той же самой клиентской программы, но приспособленной для выполнения из браузера.

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.awt.Graphics;

public class HelloApplet extends java.applet.Applet
{
    public void init() {
        try {
            // Создать и инициализировать ORB
            ORB orb = ORB.init(this, null);

            // Получить корневой контекст системы именования
            org.omg.CORBA.Object objRef =
                      orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);

            // Разрешить ссылку на объект в имя
            NameComponent nc = new NameComponent("Hello", "");
            NameComponent path[] = {nc};
            Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

            // Вызвать сервер Hello и напечатать результат
            message = helloRef.sayHello();
```

```
        } catch (Exception e) {
            System.out.println("HelloApplet exception: " + e.getMessage());
            e.printStackTrace(System.out);
        }
    }

    public void paint(Graphics g)
    {
        g.drawString(message, 25, 50);
    }

    String message = "";
}
```

Построение и выполнение программы Hello World!

Для выполнения примера необходимо зарезервировать порт выполнения. Предположим, что вы будете использовать порт TCP/IP 1050 для сервера имен Java IDL. Далее все зависит от того, где вы вознамерились выполнять этот пример (в какой операционной среде) — на Unix или в Windows NT? Если на Unix, то для доступа к портам меньше, чем 1024, вам необходимы Root-привилегии. Заметим также, что символ разделения каталогов отличается: для Unix (/) и для Windows 95 и Windows NT (\).

- Перейдите в каталог, где находится файл Hello.idl.

```
cd docs/guide/idl/examples/hello
```

- Запустите компилятор **idltojava**, чтобы создать стаб и скелетон.

Замечание

Компилятор **idltojava** использует в своей работе препроцессор. На Windows-машинах — препроцессор MS Visual C++. На Solaris машинах в каталоге /usr/ccs/lib/cpp должен находиться соответствующий препроцессор. Вы можете изменить используемый препроцессор путем установки двух переменных: `CPP` и `CPARGS`. Установите `CPP` к пути, где находится желаемый препроцессор. Установите `CPARGS` для передачи списка аргументов к препроцессору. Вы можете также выключить препроцессор добавив ключи `-fno -cpp` к командной строке idltojava.

```
idltojava  Hello.idl
```

- Скомпилируйте все java-файлы.

```
javac *.java HelloApp/*.java
```

- Убедитесь, что сервер имен выполняется.

```
tnameserv -ORBInitialPort 1050&
```

- Стартуйте сервер `HelloServer`.

```
java HelloServer -ORBInitialPort 1050
```

- Выполните клиентское приложение `HelloClient` в другой командной оболочке, чем сервер

```
java HelloClient -ORBInitialPort 1050
```

- После того как вы убедитесь, что все превосходно работает, перейдите в командное окно, где вы стартовали сервер, и уничтожьте выполняющийся процесс. Иначе он будет продолжать ждать других входящих запросов на соединение.

Усложненный пример с включением асинхронных операций

Клиентские программы часто должны реагировать на изменения или модификации, которые иногда происходят на сервере. Например, клиентская диаграмма, граф или программа электронной таблицы могли бы модифицироваться с каждым изменением цены на бирже на сервере фондового рынка. Клиент мог бы иметь две опции в этом сценарии:

- периодически запрашивать биржевую цену посредством запроса к серверу;
- просить об оповещении всякий раз, когда изменение цен происходит. Данная опция обычно именуется как `callback` (повторный вызов).

Следующий пример иллюстрирует, как клиентская программа может передавать callback-объект серверу. Сервер затем выдает запрос (method request) на callback-объект и таким образом оповещает клиента.

Этот пример идентичен предыдущему примеру Hello World, за исключением расширений callback-вызовов. Далее приводится код, необходимый для этих расширений (изменения, которые нужно внести в программу, выделены полужирным начертанием).

Объявления интерфейсов (Hello.idl)

```
module HelloApp
{
    interface HelloCallback
    {
        void callback(in string message);
    };


    interface Hello
    {
        string sayHello(in HelloCallback objRef, in string message);
    };
};
```

Метод `SayHello` изменен так, чтобы использовать объектный параметр ссылки и строковый параметр. Объектный параметр ссылки обеспечивает клиен-

та способом передачи серверу объекта callback, чтобы сервер мог его вызывать. Строковый параметр — это строка, которую сервер пошлет обратно клиенту.

Программа на сервере (HelloServer.java)

```
//

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

class HelloServant extends _HelloImplBase
{
    public String sayHello(HelloCallback callobj, String msg)
    {
       callobj.callback(msg);
       return "\nHello world !!\n";
    }
}


public class HelloServer {

    public static void main(String args[])
    {
       try{
           // create and initialize the ORB
           ORB orb = ORB.init(args, null);

           // create servant and register it with the ORB
           HelloServant helloRef = new HelloServant();
           orb.connect(helloRef);

           // get the root naming context
           org.omg.CORBA.Object objRef =
              orb.resolve_initial_references("NameService");
           NamingContext ncRef = NamingContextHelper.narrow(objRef);

           // bind the Object Reference in Naming
           NameComponent nc = new NameComponent("Hello", "");
           NameComponent path[] = {nc};
           ncRef.rebind(path, helloRef);

           // wait for invocations from clients
            java.lang.Object sync = new java.lang.Object();
            synchronized (sync) {
```

```
                sync.wait();
            }

        } catch (Exception e) {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

Программа клиента (HelloClient.java)

```
//

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

class HelloCallbackServant extends _HelloCallbackImplBase
{
    public void callback(String notification)
    {
        System.out.println(notification);
    }
}
public class HelloClient
{
    public static void main(String args[])
    {
        try{
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);

            // get the root naming context
            org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);

            // resolve the Object Reference in Naming
            NameComponent nc = new NameComponent("Hello", "");
            NameComponent path[] = {nc};
            Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));


            HelloCallbackServant helloCallbackRef =
                                        new HelloCallbackServant();
            orb.connect(helloCallbackRef);
```

```
            // call the Hello server object and print results
            String hello = hel-
loRef.sayHello(helloCallbackRef,"\ntest..\n");
            System.out.println(hello);

        } catch (Exception e) {
            System.out.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

В результате выполнения этих программ в двух различных окнах на стороне клиента получим результат, представленный на рис. 3.16.

```
Cmd.exe
HelloServer.java:26: class HelloServant is an abstract class. It can't be instan
tiated.
            HelloServant helloRef = new HelloServant();

Note: 2 files use or override a deprecated API.  Recompile with "-deprecation" f
or details.
3 errors, 1 warning

D:\Corba_Primer>idltojava -fno-cpp hello.idl

D:\Corba_Primer>javac *.java
Note: 2 files use or override a deprecated API.  Recompile with "-deprecation" f
or details.
1 warning

D:\Corba_Primer>java HelloClient -ORBInitialPort 1050

test..


Hello world !!


D:\Corba_Primer>_
```

Рис. 3.16. Результат выполнения примера Hello World с CallBack

Подробно разобрав и поняв, как сделаны эти программы, читатель книги, думается, вполне мог бы начать работу по модификации и совершенствованию первоначальной программы **Акции**, потому что возможности разработки и совершенствования программ с использованием Java и CORBA/IIOP поистине беспредельны.

Только вдумайтесь: удачно написанное Web-приложение на базе CORBA способно, без всяких переделок, к неограниченной масштабируемости. Таким приложением, без всякого риска зависания программы на сервере, могут одновременно воспользоваться десятки, а может быть, даже и сотни тысяч клиентов. С этой целью, предвидя подобные ситуации, которые с развитием Интернета и растущей глобализацией общества сегодня становятся рядовыми, и был разработан данный стандарт. Именно своей избыточной, чрезмерной усложненностью и отпугивает CORBA рядовых разработчиков. Но

попробуйте разработать Web-магазин, способный обслуживать несколько сотен одновременно посещающих его клиентов, с использованием только традиционных и ставших уже привычными технологий: JavaScript, VBScript, ASP, CGI и т. д. Вы увидите, что ваше приложение, отлично справлявшееся с работой, когда число посетителей в минуту не превышало 10, отказывается работать и блокирует всю систему, когда на ваш сервер обрушивается лавина запросов. Такой случай уже произошел в Америке и закончился полным крахом, когда на нескольких Web-серверах был выставлен на продажу электронный бестселлер Стивена Кинга "Пуля", не существующий на бумажном носителе. Ни одно из традиционных Web-приложений не смогло обработать чудовищный поток заказов. Поэтому CORBA-приложения непременно будут востребованы в ближайшем будущем наиболее крупными участниками информационного рынка.

В чем преимущества Enterprise JavaBeans (EJB)

Параллельные миры

Поскольку Интернет стал основным механизмом и первичным двигателем ведения бизнеса в XXI столетии, в борьбу за новые информационные рынки вступили гиганты. Количество и скорость распространения инноваций в сфере электронного бизнеса просто потрясает воображение. Ныне во главу угла ставится гибкость и масштабируемость разрабатываемых систем, их адаптируемость к любым изменяющимся условиям.

Сегодня имеются две ведущие сервер-ориентированные платформы:

- Sun Microsystems' Java 2 Platform, Enterprise Edition (J2EE), которая включает в себя компонентную архитектуру Enterprise JavaBeans (EJB);
- Microsoft's Windows Distributed interNet Applications Architecture (Windows DNA), которая включает COM+, реализующего сервер-компонентную архитектуру.

Java 2 Platform, Enterprise Edition (J2EE) — это не только фирменный продукт Sun, это результат большой работы кооперации ведущих фирм производителей программного обеспечения и аппаратных платформ, таких как Sun Microsystem, IBM, Oracle, BEA и др.

J2EE — это результат эволюции Java-технологий, таких как Java Database Connectivity (JDBC) и Java Remote Method Invocation (RMI). Краеугольным камнем новой технологии J2EE является Enterprise JavaBeans (EJB) — стандарт для построения современных корпоративных систем и общегосударственных систем стратегического значения.

Архитектура Windows DNA также претендует на эту роль, но, несмотря на всю мощь и интеллектуальный потенциал Microsoft, эта архитектура никогда не сможет выйти из-под опеки ее создателей и собственников. Кроме того, ее масштабы ограничены применением только одной операционной системы — Windows и только одного типа компьютеров — Intel.

Несмотря на блестящую линейку продуктов самой Microsoft и сотрудничающих с нею фирм, ориентированных на разработку приложений Интернет/интранет и систем для ведения корпоративного бизнеса, уже сейчас на этом рынке Microsoft противостоят такие великолепные системы, как:

- **IBM** ***WebSphere*** — включает в себя все лучшее из продуктов IBM *MQSeries*, *CICS* и *Encina*;
- **BEA** ***WebLogic*** — имеет громадный рынок еще до того, как Microsoft впервые предложила MTS в 1996 году;
- **Oracle** ***Oracle 8i*** — гамма продуктов, которые не нуждаются в рекомендациях;
- **Netscape** ***Netscape Application Server*** (в будущем: Sun-Netscape Alliance *iPlanet*) — основан на сервере приложений *Kiva*;
- **IONA** ***iPortal Application Server*** — основан на *OrbixWeb*, первом настоящем CORBA-продукте, в котором была сделана попытка реализовать всю грандиозную мощь этих технологий и прежде всего для Unix-ориентированного сектора IT-индустрии;
- **Inprise** ***Inprise Application Server*** — основан на *Visibroker*, является CORBA-сервером многофункционального назначения.

Корпорации должны выбирать решения, которые позволяют неограниченную масштабируемость, т. е. бесконечное наращивание мощи систем, однако именно здесь у Microsoft имеются серьезные проблемы. Windows 2000 Microsoft Datacenter Server, наиболее масштабируемая операционная система Microsoft, как сообщают, масштабирует по 16 процессоров на машину или 32 процессора для некоторых OEMS. При помощи кластеризации отдельных машин Windows 2000 может теоретически масштабировать сотни процессоров. Чтобы достигнуть такого уровня масштабируемости, основная платформа сервера должна иметь соответствующие функциональные возможности для балансировки загрузки. К сожалению, Microsoft недавно объявила, что именно этот компонент был удален из ОС-образца Windows 2000, выпущенного в феврале этого года.

Если сравнивать, то J2EE-приложения могут быть развернуты в любом разнообразии операционных систем и работать в Unix и других универсальных системах, которые могут поддерживать от 64 до 256 процессоров на сервер. Уже сейчас независимые производители ПО предлагают широкое разнообразие решений с J2EE по балансировке загрузки. Имеются стратегии для

развертывания систем, включающих десятки тысяч процессоров, приводящих к высоко масштабируемой системе.

С точки зрения надежности, Windows-ориентированные решения также уступают традиционным корпоративным системам. Ведение бизнеса в Интернете требует высокой надежности и доступности компьютеров по критерию 24×7×365. Уровень надежности должен достигать 99,999%, которую всегда обеспечивали такие системы как Sun's Solaris и IBM OS/390. Время простоя системы не должно достигать более чем 5 минут в течение года. Таковы критерии, сформулированные ведущими экспертами. Способна ли обеспечить такую высокую надежность продукция компании Microsoft, которая в момент своего возникновения, т. е. изначально позиционировала себя на рынке как производителя ПО для *персональных* компьютеров.

Краткие сведения о EJB

Для чего создана архитектура EJB

EJB задумана как архитектура и как средство для построения сложных компонентно-распределенных систем. В качестве архитектурного решения была создана программная спецификация, которая является основанием для разработки программных продуктов третьими фирмами, не исключая и саму фирму Sun — создателя и первичного публикатора этой спецификации; в качестве средства была создана специальная редакция Java и Java Development Kit (JDK), получившая название Java 2 Platform, Enterprise Edition (J2EE). J2EE включает в себя средства EJB и все необходимые API для обращения к этим средствам.

При создании спецификации были поставлены и достигнуты следующие амбициозные цели:

- архитектура Enterprise JavaBeans является стандартной компонентной архитектурой для построения распределенных объектно-ориентированных бизнес-приложений на языке программирования Java;
- архитектура Enterprise JavaBeans делает возможным построение распределенных приложений путем комбинирования компонентов, разработанных для различных платформ и операционных систем;
- архитектура Enterprise JavaBeans облегчает написание распределенных приложений: разработчикам нет необходимости знать и понимать нижние уровни системы, на которых осуществляется транспорт данных, управление транзакциями, управление многопоточностью, слежение за пулом соединений и пр.;

Замечание

Enterprise JavaBeans-приложения всегда следуют правилам WORA (Write Once, Run Anywhere): "*Написанное однажды выполняется везде*" — философии языка Java.

- архитектура Enterprise JavaBeans должна отражать все аспекты жизненного цикла программного изделия;
- архитектура Enterprise JavaBeans полностью совместима с CORBA-протоколами.

Примеры использования EJB

Если продолжать мыслить в терминах нашего, пока что весьма скромного приложения по управлению акциями, то давайте теперь представим большой брокерский дом, например Money Makers, который заказал систему на основе технологий EJB. А так оно и есть на самом деле. В статье А. Томас (Anne Thomas) (опубликована у нас в переводе Анжела Чернухо) говорится о том, что Money Makers хочет перевести своих клиентов на систему самообслуживания и намеренно поддерживать весь набор брокерских и торговых услуг посредством тонкого клиента, который связывается с серверной частью программы, реализованной с использованием CORBA/Java и EJB.

Поддержка разнообразных клиентов

Приложение для управления акциями Money Makers должно быть способно поддерживать много различных клиентских средств, включая настольные ПК, Web-браузеры, телефоны, смарт-карты и другие приспособления, использующие Интернет. Структура приложения для Money Makers, которое уже реализовано, показана на рис. 3.17.

Money Makers обслуживает миллионы клиентов и, следовательно, ее весьма волнует количество одновременно обрабатываемых запросов. Компания выбрала совместимый с EJB сервер от фирмы Big Guns System Software, состоящий из кластера на базе высокоскоростных мультипроцессоров. Система способна поддерживать прозрачное распределение и репликацию компонентов приложения по множеству серверов, т. е. обеспечивает требуемую масштабируемость при резких увеличениях нагрузки.

Повторно используемые JavaBeans

В целях экономии сил и времени компания Money Makers приобрела у фирмы Portfolio Incorporated систему управления банковскими счетами фондов акций, выполненную как набор компонентов Enterprise JavaBeans. Компания Portfolio разработала и протестировала данное приложение для Money Makers.

Механизм наследования и кастомизация

Приложение **Portfolio** управляет клиентскими счетами и подготавливает финансовые документы. Однако оно не использует в реальном времени биржевые котировки. Money Makers модернизировала компонент, оперирующий

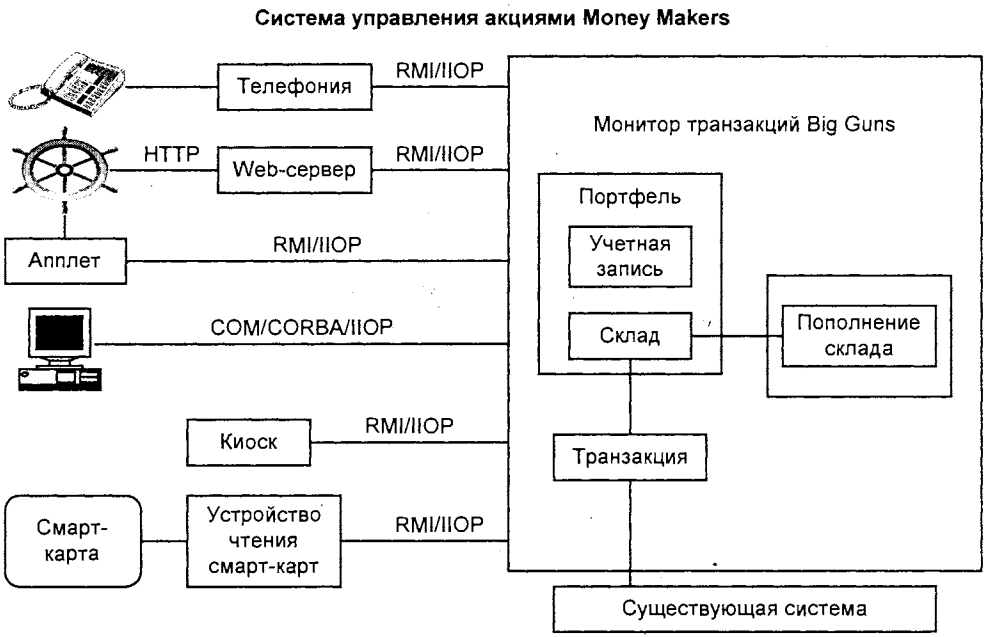


Рис. 3.17. Структура распределенного Java-приложения

акциями, чтобы он мог получать цены на акции в реальном времени. Компания и здесь нашла подходящий для этих целей компонент, выполненный в другой фирме, которую она немедленно приобрела. Для связи всех компонентов и интеграции была использована технология CORBA.

Этот реализованный в короткие сроки проект наглядно показывает все преимущества объектно-ориентированной распределенной архитектуры и возможность бесшовного включения чужих наработок в свои проекты.

Архитектура EJB

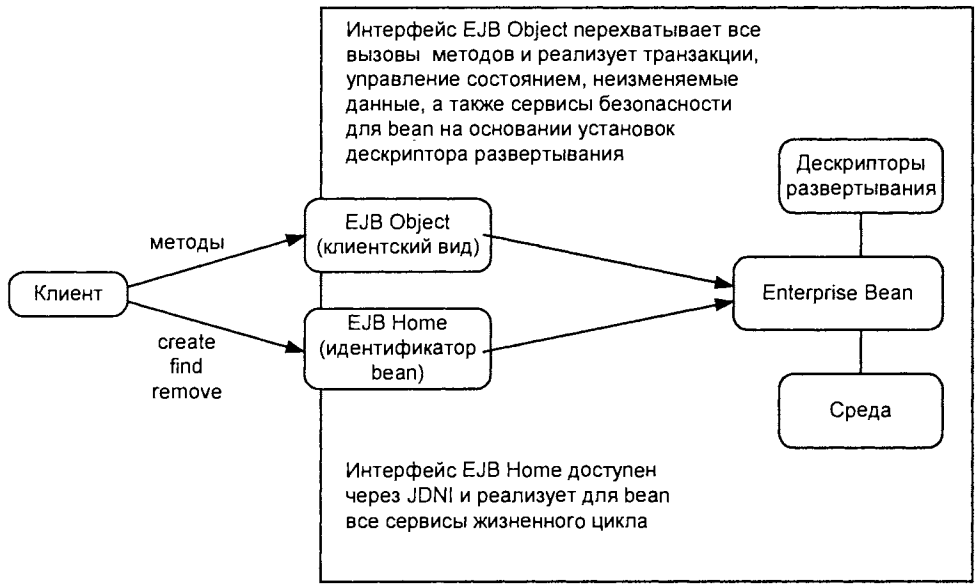
Сервер Enterprise JavaBeans

EJB-сервер предоставляет среду выполнения приложений, основанных на применении JavaBeans.

EJB-контейнер

На рис. 3.18 представлен EJB-контейнер. Сервер EJB обеспечивает один или более контейнеров EJB, предоставляющих home-интерфейсы для enterprise beans. Контейнер EJB управляет содержащимися в нем компонентами

Enterprise Beans. Для каждого компонента Enterprise Bean контейнером регистрируется объект, обеспечивается удаленный интерфейс для этого объекта, создаются и удаляются экземпляры объекта.



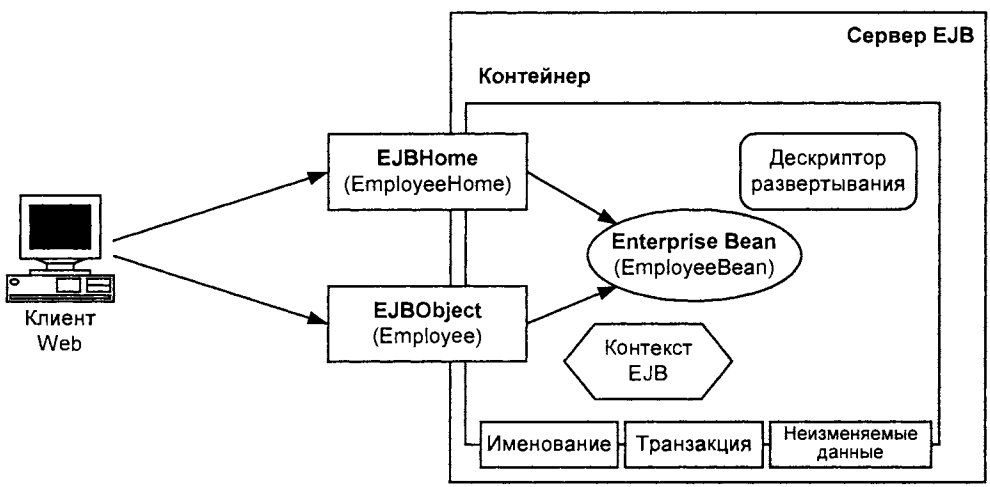


Рис. 3.18. Контейнер Enterprise Beans

Инсталляция Enterprise Beans в EJB-контейнере

Любое количество классов EJB можно упаковать в один EJB-контейнер. Физическая реализация контейнера EJB не определяется спецификацией Enterprise JavaBeans. Контейнер EJB может быть реализован как обычный многопоточный процесс внутри сервера EJB. Конечно, программисты могут создавать на Java свои собственные программные контейнеры, но, как правило, этого делать не нужно. Крупнейшими производителями EJB-контейнеров и EJB-серверов уже стали всем известные фирмы, которые поспешили позиционировать свои продукты в качестве EJB-серверов. К ним относятся:

- BEA WebLogic WebLogic Server
- Forte SynerJ
- GemStone GemStone/J 3.0
- IBM WebSphere 3.0
- Inprise Application Server
- Luna Server Netscape Application Server 4.0
- Oracle J Server (включен в Oracle 8i или Oracle 8.1.5)
- Persistence PowerTier for EJB 5.03
- SilverStream Application Server
- Sun NetDynamics5.0
- Sybase Jaguar CTS

Как работает EJB

Главной задачей EJB-сервера является планирование ресурсов и координация работы приложений, использующих технологию Enterprise JavaBeans. EJB-сервер должен содержать один или более контейнеров с объектами в виде Beans, которые могут распределенно вызываться и использоваться, никуда при этом не перемещаясь и оставаясь на сервере. Контейнер отвечает за координацию транзакций, регистрацию каждого содержащегося в нем компонента и обеспечение удаленных интерфейсов для требуемых объектов, доступ к которым осуществляется по создаваемым объектным ссылкам.

Дополнительно контейнер может управлять персистентными (постоянными, неизменяемыми) данными внутри объекта. Любое количество классов может быть инсталлировано внутри одного контейнера. Соответствующий класс Beans назначается всегда одному и только одному EJB-контейнеру. Физическая спецификация EJB-контейнера никак не определяется в спецификации EJB, это может быть многопотоковый процесс внутри EJB-сервера. Схематично процесс упаковки Beans в контейнеры, а контейнеров на EJB-сервер показан на рис. 3.19.

Рис. 3.19. Сервер и контейнер EJB

EJB-контейнер действует как транспортный механизм между клиентом и Enterprise Bean. В момент развертывания приложения контейнер генерирует EJB.

Home-интерфейс для представления классов и EJB Object-интерфейс для каждого экземпляра Bean. EJB Home-интерфейс (в виде Bean-идентификатора) доступен через стандартное средство Java (JNDI — Java naming and directory) и идентифицирует Bean-класс, который используется для создания, нахождения и удаления экземпляров объекта данного класса. Интерфейс EJB Object (как представление для клиента) обеспечивает доступ к бизнес-алгоритмам, которые реализует выбранный Bean. Все клиенты должны запрашивать эти интерфейсы. Клиентские запросы перехватываются EJB-контейнером, который координирует транзакции и прочие операции взаимодействия объектов. Таким образом, EJB — это целостная концепция распределенного объектного управления, включающая в себя все необходимые механизмы, в том числе и транспортные, и сетевые, и транзакционные.

Клиент

Клиентская часть EJB обеспечивается двумя интерфейсами:

- *Home Interface*, который обеспечивает методы для создания, удаления и обнаружения объектов в контейнере. Клиент может использовать JNDI для поиска имен соответствующих классов;
- *Remote Interface*, который определяет бизнес-методы, предлагаемые выбранным Bean.

Bean

EJB-классы и экземпляры классов — а коротко Beans (Бобы) — осуществляют три категории содержащихся в них и дистанционно вызываемых методов:

- методы для создания, нахождения и доступа к экземплярам объектов посредством home-интерфейса;
- методы, поддерживающие бизнес-алгоритмы посредством remote-интерфейса;
- методы интерактивного взаимодействия внутри контейнера.

Замечание

Эти методы скрыты для клиентов.

Изменяемые и неизменные объекты

Технология Enterprise JavaBeans поддерживает как изменяемые (transient), так и неизменные (persistent) объекты. Изменяемые объекты называются *сеансовыми компонентами Bean* (session beans), т. е. те, которые существуют только на время сеанса работы с ними; а неизменные объекты называются *сущностными компонентами Bean* (entity beans), т. е. они могут существовать долгое время и оставаться неизменными от сеанса к сеансу.

Сеансовые компоненты Bean (Session Beans)

Данный тип компонента создается клиентом и существует только во время одного сеанса. Он выполняет операции, необходимые клиенту, связанные с доступом к базе данных или с вычислениями. Session Beans могут поддерживать транзакции, хотя, как правило, они не могут быть восстановлены в случае сбоя системы. Могут также поддерживать коммуникации посредством стандартных методов и транзакций. Контейнер управляет коммуникационным состоянием этого компонента, если требуется выгрузить его из памяти. Сеансовый Bean сам управляет своими собственными данными.

Сущностные компоненты Bean (Entity Beans)

Данный тип компонента — это объектное представление неизменяемых данных, содержащихся в базе данных. Первичный ключ определяет каждый экземпляр Entity Bean. Сущностные компоненты Bean могут создаваться либо путем включения данных непосредственно в базу данных, либо путем создания объекта (с помощью метода `Object Factory Create`). Они поддерживают транзакции и могут восстанавливаться в случае сбоя системы.

Процесс обслуживания различных клиентов в EJB можно представить в виде, показанном на рис. 3.20.

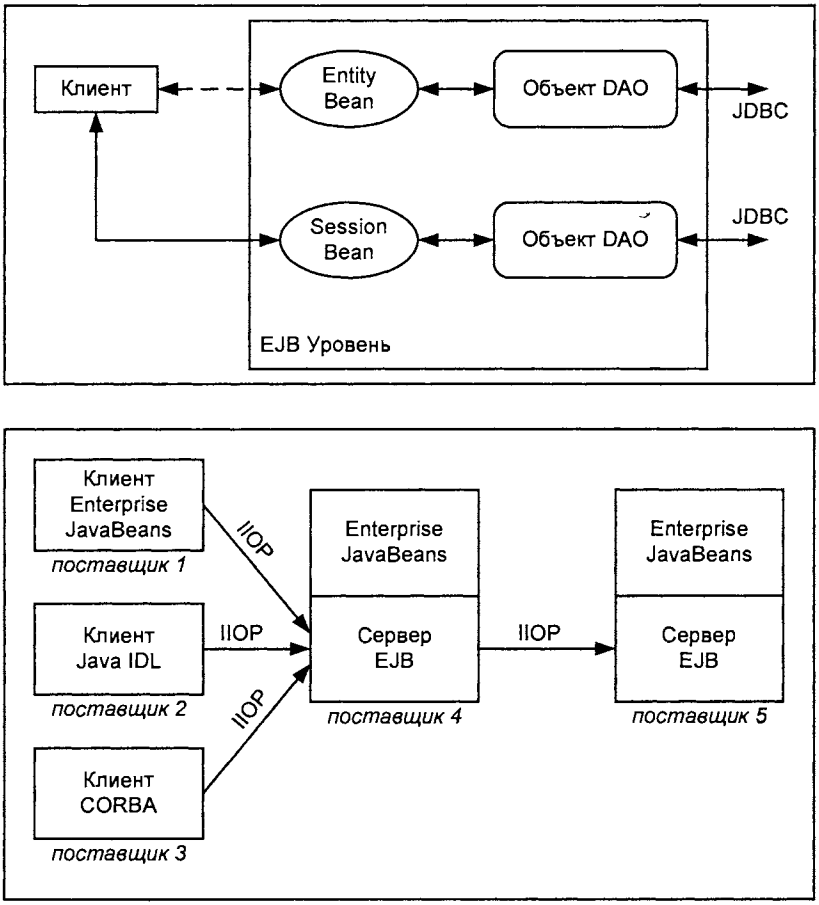


Рис. 3.20. Процесс обслуживания клиентов в EJB

Иногда возникает вопрос — *а чем EJB отличается от обычного JavaBean, который мы разбирали в предыдущей главе*? С точки зрения сущности — ничем, а с точки зрения построения и способов использования — многим. В узком смысле можно сказать, что архитектура EJB логически расширяет возможности компонентной модели JavaBean так, чтобы иметь возможность поддерживать компоненты для серверов.

Если брать более широко, то можно утверждать, что EJB — это концепция многомашинных распределенных архитектур, включающая в себя все требуемые звенья. Это самодостаточная система, которая содержит в себе все необходимое. *А что надо, чтобы перейти от простого JavaBean к EJB?* Компонентная модель EJB-сервера определяет набор стандартных, независимых от производителя, интерфейсов для всех серверов Java-приложений. Приложения класса enterprise (промышленные приложения) требуют доступа к

различным инфраструктурным сервисам: распределенным процессам, коммуникациям, работы с деревом каталогов, управления транзакциями, обмена сообщениями, доступа к данным, персистентности, а также возможности совместного использованием ресурсов. Данные инфраструктурные сервисы часто реализуются на различных платформах посредством различных продуктов и технологий, что вызывает трудности при создании переносимых систем приложений класса enterprise. Возможности переносимости EJB-компонентов становятся реальными благодаря платформе Enterprise Java (Java2EE).

Jar-файл EJB

Этот файл представляет собой специальный архив, который переносится на каждую машину, где необходимо использовать EJB-функции. Данный архив может быть набором enterprise Beans или готовой прикладной системой. Jar-файл EJB включает информацию о содержимом данного файла, о классах и их описаниях, требуемых для управления ими со стороны EJB-контейнера. Для создания таких архивов в J2EE SDK существует специальное средство в виде Java-архиватора.

Чем Enterprise JavaBeans отличаются от сервлетов

Сервлеты своим названием говорят о том, что они являются сервер-ориентированными эквивалентами апплетов. Они отличаются от апплетов тем, что не используют графический интерфейс типа Abstract Window Toolkit (AWT) или Swing. Главное их предназначение — вытеснить часто неудобные программы Common Gateway Interface (CGI).

Основное преимущество java-сервлетов — это их платформенная независимость и защита, которую они обеспечивают. Java-сервлеты являются функциональным расширением Web-серверов. Преимущество EJB-компонентов заключается в возможности их многократного использования. EJB-компоненты проще программировать, чем сервлеты, поскольку множество функций типа управления транзакциями, защитой, соединениями и т. д. вынесены за пределы Beans. Кроме того, любое приложение на основе Java может использовать комбинацию сервлетов, JSPS и EJB.

Краткие сведения о J2EE

Теперь EJB является частью J2EE. Если проследить эволюцию развития Java-технологий с момента их зарождения в середине 90-х годов, то голова пойдет кругом. Первое, что приходит на ум — никакая это не эволюция, а революция! Такого обилия идей и стандартов я, например, в своей жизни еще не видел. Если раньше можно было обойтись простеньким JDK, то теперь весь набор Java-технологий исчисляется многими десятками. Слава

богу, медленно, но верно, они стали группироваться в законченные системы. Java 2 Enterprise Edition (J2EE) — пример такой мощной системы. J2EE включает в себя:

- **JDBC 2.0 Extension.** *JDBC 2.0 Standard Extension API* (JDBC extension specification). Copyright 1998, 1999, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/jdbc/jdbc20.stdext.pdf**.
- **JTA 1.0.** *Java Transaction API, Version 1.0.1* (JTA specification). Copyright 1998, 1999, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/jta/index.html**.
- **JMS.** *Java Message Service, Version 1.0.1* (JMS specification). Copyright 1998, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/jms /jms-101-spec.pdf**.
- **JNDI 1.2.** *Java Naming and Directory Interface Specification, Version 1.2* (JNDI specification). Copyright 1998, 1999, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/jndi/1.2/javadoc/index.html**.
- **Servlet 2.2.** *Java Servlet Specification, Version 2.2* (Servlet specification). Copyright 1998, 1999, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/servlet/2.2/index.html**.
- **JSP 1.1.** *JavaServer Pages Specification, Version 1.1* (JSP specification). Copyright 1998, 1999, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/jsp/techinfo.html**.
- **EJB 1.1** *Enterprise JavaBeans Specification, Version 1.1* (EJB·specification). Copy-right 1998, 1999, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/ejb/docs.html**.
- **RMI-IIOP**
- **JavaMail** *JavaMail API Specification, Version 1.1* (JavaMail specification). Copy-right 1998, Sun Microsystems, Inc. Доступен на **http://java.sun.com/products/javamail/index.html**.
- **JAF 1.0.** *JavaBeans Activation Framework Specification, Version 1.0.1* (JAF speci-fication). Copyright 1998, Sun Microsystems, Inc. Доступен на **http://java.sun.com/beans/glasgow/#activation**.

Компонентные технологии

Типы компонентов из архитектуры J2EE и их контейнеров проиллюстрированы на рис. 3.21.

Например, из рисунка видно, что в Web-контейнере возможно прямое использование API Java Standart Edition для обращения к JSP и сервлетам, но из EJB-контейнера это напрямую сделать нельзя. Также из апплета нельзя напрямую использовать интерфейсы управления транзакциями — JTA.

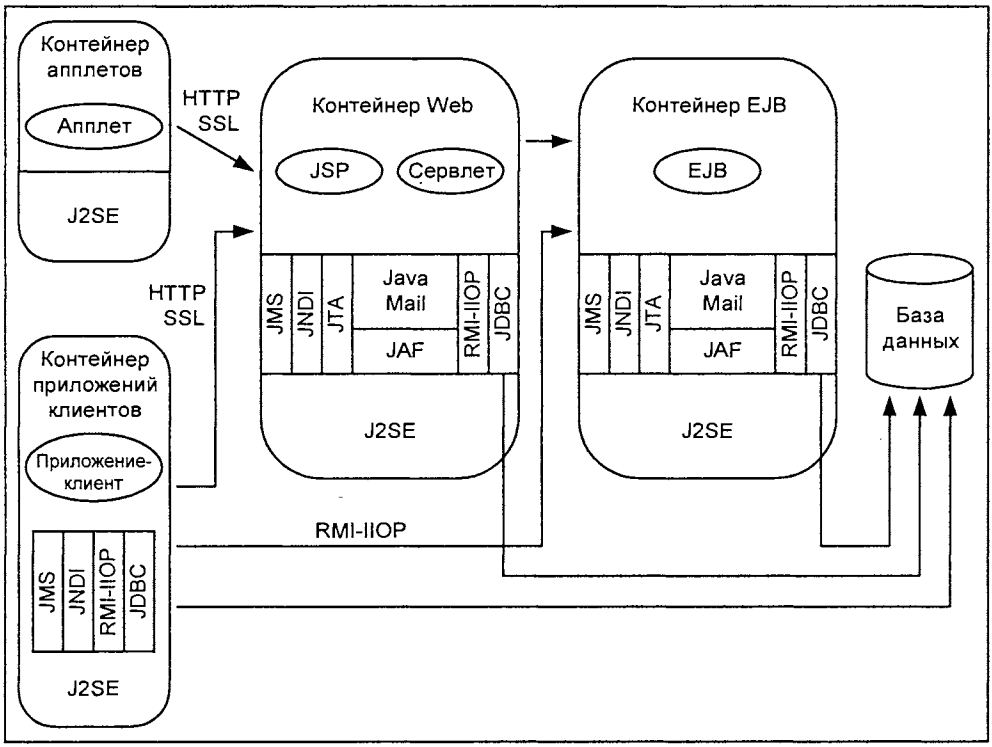


Рис. 3.21. Взаимосвязь компонентов архитектуры J2EE

Технологии обслуживания (Service Technologies)

Эти J2EE-технологии позволяют приложениям получать доступ к многочисленным сервисам различного функционального назначения — работе с базами данных, управлению транзакциями, каталогам имен и пр.

JDBC

JDBC — это набор программных интерфейсов (API) для доступа к разнообразным базам данных через источники OBBC или самостоятельные JDBC-драйверы. При этом обеспечивается:

- загрузка и конфигурирование драйверов на клиентах;
- выполнение соединений с сервером и аутентификация;
- управление транзакциями;
- пересылка SQL-операторов к СУБД-машине;
- получение результатов выполнения и их интерпретация.

Java Transaction API and Service

Сервер управления транзакциями (JTA) позволяет организовать работу с СУБД наиболее эфффективным способом.

Java Transaction Service (JTS) — это специально разработанная спецификация для JTA, которая включает в себя все механизмы управления транзакциями, такими как управление транзакционным пулом, демаркация, синхронизация и т. д.

Java Naming and Directory Interface

Java Naming and Directory Interface (JNDI) является API, который обеспечивает работу с глобальными каталогами и серверами поддержки имен.

Поскольку JNDI является системно-независимым, его можно использовать для доступа и манипулирования объектами поддержки имен для таких популярных систем, как LDAP, NDS, DNS и NIS.

Будущая версия J2EE-платформы будет поддерживать архитектуру специальных коннекторов для взаимодействия с любыми EIS (Enterprise Information Systems), такими как ERP-системы управления транзакциями и серверы СУБД. Подобные коннекторы будут встроены практически в любой EJB-сервер.

Java IDL

Java IDL позволяет Java-клиентам использовать объекты CORBA, которые были определены при помощи IDL для любых языков программирования, имеющих CORBA-отображение. Java IDL является частью платформы J2SE.

RMI-IIOP

RMI-IIOP — есть реализация RMI API поверх IIOP. RMI-IIOP позволяет программировать удаленные интерфейсы взаимодействия объектов на Java. Удаленный интерфейс может быть преобразован в IDL и выполнен на любом другом языке, который обеспечивается CORBA-отображением.

RMI-IIOP содержит компилятор **rmic**, который генерирует:

- стабы для клиента и сервера, способные работать с любым ORB;
- IDL-файл, совместимый с RMI-интерфейсом;
- CORBA API и ORB.

Технологии управления сообщениями (Messaging Technologies)

Технологии управления сообщениями являются одними из старейших программных технологий, который позволяют организовать интенсивный программный обмен между задачами, выполняющихся на разных компьютерных платформах в разных операционных системах.

Java Message Service

Java Message Service (JMS) — это API для систем управления сообщениями или для использования уже существующих систем, таких как IBM MQ Series, TIBCO Rendezvous и др.

JavaMail

JavaMail API обеспечивает набор абстрактных классов и интерфейсов, которые включают систему электронной почты. Абстрактные классы и интерфейсы поддерживают много различных форматов и протоколов.

Абстрактные классы в JavaMail могут иметь подклассы, чтобы обеспечить новые протоколы и новые функциональные возможности, когда это необходимо. Кроме того, JavaMail включает конкретные подклассы для реализации широко применяемых протоколов почты Интернета и по спецификациям RFC822 и RFC2045. Разработчики могут использовать классы JavaMail, чтобы обеспечить такие спецификации и протоколы, как IMAP4, POP3 и SMTP.

J2EE — открытая платформа разработок на Java

Рис. 3.22 дает графическое представление многоуровневой среды функционирования J2EE. На клиентском уровне могут существовать два подуровня специализированного сервера защиты.

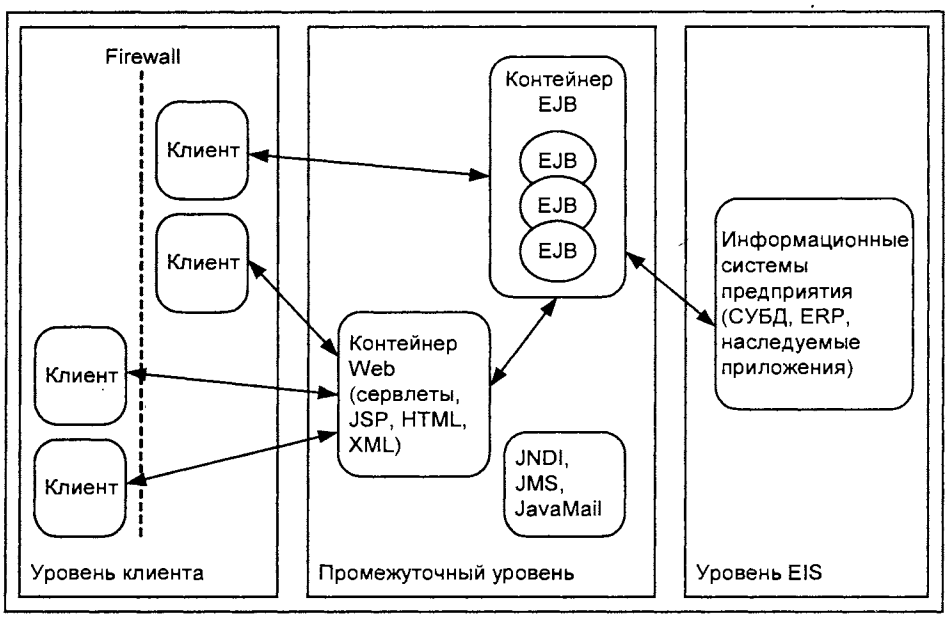


Рис. 3.22. Многоуровневая среда функционирования J2EE

Уровень клиента

Во многих случаях обслуживание обеспечивается непосредственно из браузера. Это может быть сделано путем генерации HTML в среднем уровне, при использовании JSP-технологии (о ней — позже).

Когда обслуживание включает более сложные задачи, то часто бывает необходимо выполнять некоторые функциональные возможности непосредственно на клиенте. Эти функциональные возможности обычно выполняются как компоненты JavaBeans, которые взаимодействуют с системой в среднем уровне через сервлеты (servlets) *(см. главу 4)*.

Обычно на первом уровне компоненты JavaBeans загружаются и выполняются в виде апплетов, как небольшие приложения, которые загружаются автоматически в браузер. При этом могут возникнуть некоторые проблемы, связанные с использованием устаревших версий JVM, встроенных в некоторые типы браузеров. Может внести свои проблемы также и OS-специфика, связанная с выполнением некоторых Java-программ.

В случае использования сервлетов с переносом обслуживания на средний уровень, клиенты этой категории оказываются в более выгодном положении, потому что они применяют только стандарт HTTP, существующий в неизменном виде везде, в любой операционной системе.

Средний уровень (уровень сервера приложений)

В фокусе модели программирования J2EE находится именно средний уровень.

Функции среднего уровня выполнены как компоненты Enterprise JavaBeans, что позволяет корпоративному разработчику концентрировать свои усилия на реализации деловой логики, сбрасывая все остальные проблемы, связанные с коммуникациями и конвергенцией информации на EJB-сервер и JSP-технологию. JSP-технология и сервлеты облегчают написание Web-интерфейсов и всего остального, связанного с представлением информации для клиета в соответствии с его запросами и предпочтениями.

Уровень информационных систем (EIS Tier)

Средний уровень взаимодействует с корпоративными базами данных и ERP-системами через стандартный набор интерфейсов и API:

- JDBC — доступ к реляционным базам данных из Java;
- JNDI (Java Name and Directory Interface) — API для доступа к ERP-системам и глобальным каталогам;
- Java Message Service (JMS) — API для систем управления сообщениями, подобных IBM MQ Series и TIBCO Rendez-vous;
- JavaMail — API для получения и отправки электронной почты;
- Java IDL — API для вызова CORBA-сервисов.

В добавление к этому специализированный доступ к Enterprise Resource Planning (ERP) системам и мэйнфреймам, таким как IBM's CICS и IMS обеспечивается при помощи J2EE-коннекторов. Так как каждая из этих систем обладает повышенной сложностью, они требуют уникальных инструментальных средств и поддержки. Enterprise beans объединяют использование объектов доступа к ним через коннекторы и сервисные API с деловой логикой, реализуемой на среднем уровне.

Сценарии приложений

В спецификации J2EE предусмотрено большое количество сценариев различных приложений, которые все могут использоваться. Действительно, спецификации имеют тенденцию к тому, чтобы охватывать и поощрять, как можно большее богатство и многообразие. Решения на уровне приложения могут быть самыми разнообразными. На рис. 3.23 изображены некоторые ключевые сценарии, по которым может быть построено распределенное приложение, включающее либо Web-контейнер, либо EJB-контейнер, либо оба контейнера.

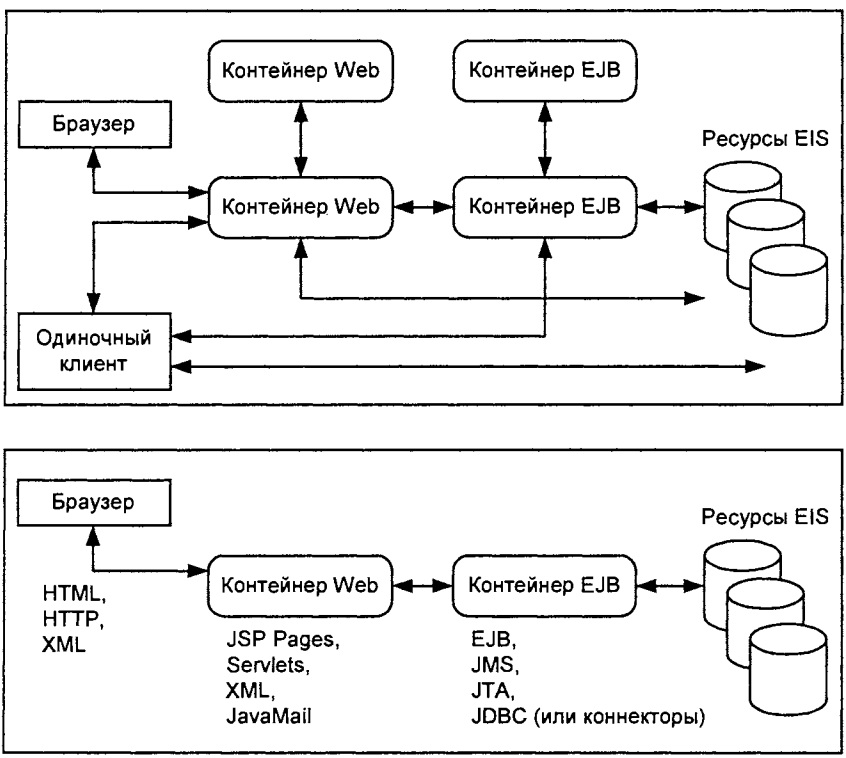


Рис. 3.23. Сценарии разработки распределенного приложения

Как J2EE стыкуется с EJB

J2EE-платформа и EJB-архитектура

J2EE-платформа и EJB-архитектура имеют почти одинаковые и, в то же время, отличающиеся цели. Из J2EE-перспективы компонентная модель EJB представляется становым хребтом программной модели J2EE. И, наоборот, из EJB-перспективы компонентная платформа J2EE лишь дополняет EJB-спецификацию тем, что:

- специфицирует APIs, которые разработчики EJB могут использовать в своих целях;
- более широко определяет распределенную программную среду, в которой enterprise beans могут быть использованы для построения и реализации бизнес-логики.

Примеры использования

Говорят, что лучше один раз увидеть, чем сто раз услышать. Разберем стандартный пример с сайта **developer.java.sun.com**, основанный на использовании J2EE SDK.

J2EE SDK — это некоммерческое расширение J2EE-платформы, совершенно свободно распространяемое фирмой SUN в целях демонстрации, прототипирования и обучения. J2EE SDK приходит с J2EE-сервером приложений, Web-сервером, базой данных и J2EE APIs, а также специальными средствами разработки и развертывания приложений.

В качестве примера создадим программу ультратонкого клиента, выполняющую сервлет, который не реализует никакой деловой логики, а вместо этого использует сеансовый Bean (session Bean), который производит некоторые простые вычисления по схеме, показанной на рис. 3.24.

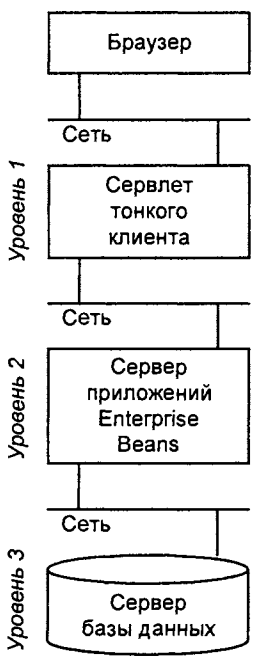


Рис. 3.24. Схема реализации примера J2EE

Для выполнения примера вам потребуются Java 2 SDK Enterprise Edition (J2EE) Version 1.2 Release и Java 2 SDK Standard Edition. Пусть они будут находиться в каталогах d:\jdk1.2 и d:\jdk1.2_sdk соответственно.

В системном **PATH** должен быть прописан путь к подкаталогу **bin** этих каталогов.

В переменной **CLASSPATH** должно присутствовать: d:\jdk1.2_sdk\lib\j2ee.jar.

Пользовательская HTML-страница — это обычный файл Bonus.html, который через стандартную форму получает от пользователя два поля данных: номер счета социального обеспечения и коэффициент. Когда пользователь нажимает кнопку **Submit**, BonusServlet отыскивает данные конечного пользователя в базе данных и передает данные пользователя к сеансовому Bean. Сеансовый Bean вычисляет сумму и возвращает значение суммы в Servlet, который затем возвращает другую HTML-страницу со значением полученной суммы для конечного пользователя, чтобы он ее посмотрел.

Схема прохождения данных показана на рис. 3.25.

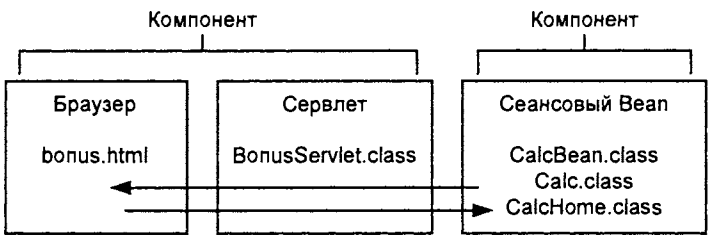


Рис. 3.25. Схема прохождения данных

Исходный код HTML-страницы

```
<HTML>
<BODY BGCOLOR = "WHITE">
<BLOCKQUOTE>
<H3>Bonus Calculation</H3>
<FORM METHOD="GET"
ACTION="BonusAlias">
<P>
Enter social security Number:
<P>
<INPUT TYPE="TEXT" NAME="SOCSEC"></INPUT>
<P>
Enter Multiplier:
<P>
<INPUT TYPE="TEXT" NAME="MULTIPLIER"></INPUT>
<P>
```

```
<INPUT TYPE="SUBMIT" VALUE="Submit">
<INPUT TYPE="RESET">
</FORM>
</BLOCKQUOTE>
</BODY>
</HTML>
```

Исходный код сервлета

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import Beans.*;

public class BonusServlet extends HttpServlet {
CalcHome homecalc;
Calc theCalculation;

public void init(ServletConfig config)
throws ServletException{

// Просмотр интерфейса home
try{
InitialContext ctx = new InitialContext();
Object objref = ctx.lookup("calcs");
homecalc =
(CalcHome)PortableRemoteObject.narrow(
objref,
CalcHome.class);
} catch (Exception NamingException) {
NamingException.printStackTrace();
}
}
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
{
String socsec = null;
int multiplier = 0;
double calc = 0.0;
PrintWriter out;
response.setContentType("text/html");
String title = "EJB Example";
out = response.getWriter();
```

```
out.println("<HTML><HEAD><TITLE>");
out.println(title);
out.println("</TITLE></HEAD><BODY>");
try{
// Извлечение информации Bonus и Social Security
String strMult =
request.getParameter("MULTIPLIER");
Integer integerMult = new Integer(strMult);
multiplier = integerMult.intValue();
socsec = request.getParameter("SOCSEC");
// Подсчет bonus
double bonus = 100.00;
theCalculation = homecalc.create();
calc =
theCalculation.calcBonus(multiplier, bonus);
}catch(Exception CreateException){
CreateException.printStackTrace();
}
// Отображение данных
out.println("<H1>Bonus Calculation</H1>");
out.println("<P>Soc Sec: " + socsec + "<P>");
out.println("<P>Multiplier: " + multiplier + "<P>");
out.println("<P>Bonus Amount: " + calc + "<P>");
out.println("</BODY></HTML>");
out.close();
}
public void destroy() {
System.out.println("Destroy");
}
}
```

Комментарии к сервлету

Метод `init`

Метод `BonusServlet.init` определяет собственный местный (домашний) интерфейс (home interface) сеансового Bean и создает экземпляр объекта. Этот метод использует имя JNDI, специфицированное в момент сборки компонента (`calcs`), чтобы получить ссылку к интерфейсу по имени. Следующая строка передает ссылку и класс `home interface` методу объекта `PortableRemoteObject.narrow`, для того, чтобы привести ссылку к типу `CalcHome`.

```
InitialContext ctx = new InitialContext();
Object objref = ctx.lookup("calcs");
homecalc = (CalcHome)PortableRemoteObject.narrow(objref, CalcHome.class);
```

Метод doGet

Список параметров для метода doGet берется из объекта запрос/ответ. Браузер посылает запрос к сервлету и он присылает ответ обратно браузеру. Реализация метода использует информацию из объекта-запроса, чтобы выяснить, кто сделал запрос, каков тип запроса, какие заголовки HTTP были посланы, и использует объект-ответ, чтобы создать HTML-страницу в ответ на запрос браузера.

Метод doGet использует исключительную ситуацию IOEXCEPTION, если случаются проблемы с вводом/выводом, и ServletException, если запрос не может быть обработан. Чтобы вычислить значение денежной суммы, которую должен получить пользователь, метод doGet создает home-интерфейс и вызывает метод calcBonus.

Создание сеансового Bean

Сеансовый Bean (Session Bean) осуществляет как бы промежуточное (transient conversation) взаимодействие с клиентом. Если сервер или клиент дают сбой, данные теряются. Напротив, в случае использования сущностных компонентов Bean данные сохраняются в базе данных.

Поскольку наш Bean выполняет относительно простые вычисления по запросу из BonusServlet и может быть повторно инициализирован в случае аварийного отказа, нам можно использовать именно сеансовый Bean в этом примере.

Рис. 3.26 графически иллюстрирует процесс прохождения данных.

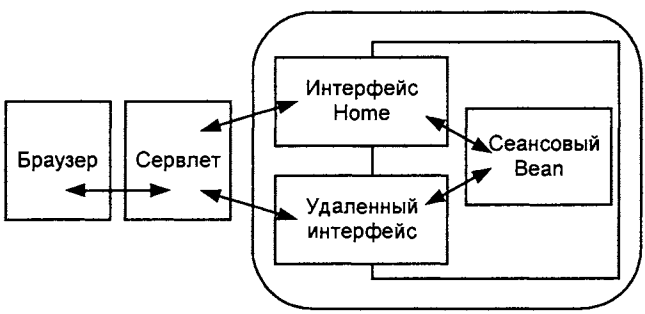


Рис. 3.26. Bean-контейнер

Компоненты приложения

CalcHome

Сервлет BonusServlet не работает непосредственно с сеансовым Bean, но содает экземпляр home-интерфейса, который расширяет EJBHome и имеет метод create для создания сеансового Bean в его контейнере.

Метод `CreateException` используется в случае, если сеансовый Bean не может быть создан, а метод `RemoteException` применяется только в результате ошибок коммуникаций.

```
package Beans;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface CalcHome extends EJBHome {
Calc create() throws CreateException,
RemoteException;
}
```

Calc

Когда home-интерфейс создан, сервер приложений J2EE создает удаленный (remote) интерфейс и сеансового Bean. Удаленный интерфейс расширяет объект `EJBObject` и определяет метод `calcBonus` для подсчета значения денежной суммы. Этот метод требует обязательного использования метода `javax.rmi.RemoteException` и выполняется классом `CalcBean`.

```
package Beans;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Calc extends EJBObject {
public double calcBonus(
int multiplier, double bonus)
throws RemoteException;
}
```

CalcBean

Сеансовый Bean, говоря простыми словами, реализует интерфейс `SessionBean` и обеспечивает выполнение метода `calcBonus`. Методы `SetSessionContext` и `ejbCreate` вызываются контейнером после `BonusServlet`.

```
package Beans;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class CalcBean implements SessionBean {

public double calcBonus(int multiplier,
double bonus) {
```

```
double calc = (multiplier*bonus);
return calc;
}
public void ejbCreate() { }
public void setSessionContext(
SessionContext ctx) { }
public void ejbRemove() { }
public void ejbActivate() { }
public void ejbPassivate() { }
public void ejbLoad() { }
public void ejbStore() { }
}
```

Компиляция сеансового Bean

На Unix

```
#!/bin/sh
cd /home/monicap/J2EE
J2EE_HOME=/home/monicap/J2EE/j2sdkeel.2
CPATH=.:$J2EE_HOME/lib/j2ee.jar
javac -d . -classpath "$CPATH" Beans/CalcBean.java
Beans/CalcHome.java Beans/Calc.java
```

В Windows

```
cd d:\mybook
cd examples\J2EE
set J2EE_HOME=d:\jdkl.2_sdk
set CPATH=.;%J2EE_HOME%\lib\j2ee.jar
javac -d . -classpath %CPATH% Beans/CalcBean.java
Beans/CalcHome.java Beans/Calc.java
```

Компиляция сервлета

На Unix

```
cd /home/monicap/J2EE/ClientCode
J2EE_HOME=/home/monicap/J2EE/j2sdkeel.2
CPATH=.:$J2EE_HOME/lib/j2ee.jar:
/home/monicap/J2EE
javac -d . -classpath "$CPATH" BonusServlet.java
```

В Windows

```
cd examples\J2EE\ClientCode
set J2EE_HOME=examples\J2EE\jdkl.2_sdk
```

```
set CPATH=.;%J2EE_HOME%\lib\j2ee.jar;
examples\J2EE
javac -d . -classpath %CPATH% BonusServlet.java
```

Старт сервера приложений J2EE

В состав Java 2 SDK, как уже говорилось, входит и свой сервер приложений J2EE, и своя DBMS (СУБД) Cloudscape, и другие средства для развертывания распределенных приложений на основе EJB. Чтобы стартовать сервер приложений, необходимо выполнить следующую команду:

```
j2ee -verbose
```

Переменные, указывающие каталоги, где инсталлированы JDK1.2 и Java 2 SDK, должны быть заданы в переменных:

```
J2EE_HOME
JAVA_HOME
```

На рис. 3.27 показан старт сервера приложений J2EE.

```
Сеанс MS-DOS - j2ee -verbose
04.12.99  19:55                    726 realntool.bat
04.12.99  19:55                  1 012 runclient.bat
04.12.99  19:55                    904 setenv.bat
04.12.99  19:55                    462 userconfig.bat
04.12.99  19:55                    945 verifier.bat
              13 File(s)        10 376 bytes
                            207 101 952 bytes free

D:\jdk1.2_sdk\bin>j2ee -verbose

Naming service started: :1050
Published the configuration object ...
Binding DataSource, name = jdbc/Cloudscape, url = jdbc:cloudscape:rmi:Cloudscape
DB;create=true
Configuring web service using "default"
Web service started: :9191
Web service started: :7000
Configuring web service using "default"
Web service started: :8000
Endpoint [SSL: ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=7000]] shutdow
n due to exception: javax.net.ssl.SSLException: No available certificate corresp
onds to the SSL cipher suites which are enabled.
endpoint down: :7000
J2EE server startup complete.
```

Рис. 3.27. Старт сервера приложений J2EE из состава Java 2 SDK

Замечание

При запуске J2EE не рекомендуется использовать `HotSpot`.

Запуск среды развертывания приложения

Чтобы запустить скомпилированное и построенное вами приложение в среде Java 2 SDK, вы должны подготовить его к выполнению, для чего служат специальные графические средства, написанные на Java. Запуск графиче-

ской среды для развертывания вашего приложения осуществляется по команде:

d:\jdk1.2_sdk\bin\deploytool

Программа Deploytool имеет четыре основных окна. Окно **Local Applications** отображает J2EE-приложения и их компоненты, а окно **Inspecting** — информацию относительно выбранного приложения и компонентов. Окно **Servers** сообщает, где сервер приложения выполняется — на локальной или на главной ЭВМ. И окно **Server Applications** сообщает — какие приложения были установлены.

Компоновка распределенного приложения

Прежде чем логически соединять воедино распределенное приложение, необходимо создать специальный архивный файл (Enterprise Archive — EAR). В нашем примере имеется два типа архивных файлов:

- файл Web Archive (WAR), который содержит файлы bonus.html и BonusServlet;
- файл Java Archive (JAR), который содержит три Bean-файла: Calc.class, CalcHome.class и CalcBean.class.

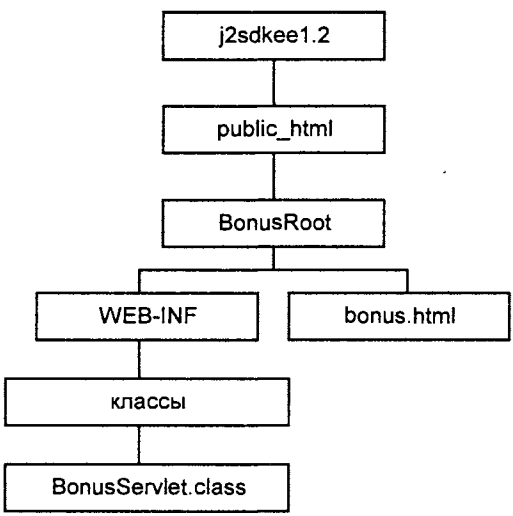


Рис. 3.28. Структура приложения

Чтобы построить EAR-файл, нужно:

- создать JAR-файл сеансового Bean (CalcBean.jar);
- создать WAR-файл Web-компонентов (Bonus.war);
- дать имя JNDI для сеансового Bean (`calcs`).

Из меню **File** выберите пункт **New Application**. Введите **BonusApp** в качестве имени. Укажите каталог, где вы желаете разместить EAR-файл. При помощи кнопки **Add** добавьте желаемые файлы, правильно выбирая пункты меню, либо **New Enterprise Bean** — для .jar-файлов, либо **New Web Component** — для WAR-файлов. Затем присвойте JNDI-имя и сформируйте Root Context так, как это показано на рис. 3.28.

Можно еще воспользоваться меню **Tools** для проверки скомпонованного приложения с целью обнаружения в его структуре каких-либо нестыковок.

Выполнение распределенного приложения J2EE

Встроенный в Java 2 SDK Web-сервер после запуска J2EE выполняется на порту 8000 по умолчанию. Чтобы открыть нужную нам HTML-страницу, укажите в браузере следующий URL-адрес:

http://localhost:8000/BonusRoot/bonus.html

Заполните появившуюся на экране форму и нажмите кнопку **Submit**. Все должно сработать. Вы вскоре увидите на экране что-нибудь вроде этого:

```
Bonus Calculation

Soc Sec: 777777777
Multiplier: 25

Bonus Amount 2500.0
```

Цель достигнута — распределенные части вашего приложения работают совместно под управлением Java 2 Enterprise Edition и используют ключевые компоненты распределенной архитектуры, такие как EJB, Servlets и JDBC. Именно на таких же принципах строятся сегодня и другие грандиозные приложения по объему, сложности и возможностям далеко превосходящие это простенькое приложение, но идеологически родственные ему, — такие как Oracle 8i, IBM WenSphere и Inprise Application Server и др.

Приложение к главе 3. Интернет-ресурсы и источники

- Технологии WWW, CORBA и Java в построении распределенных объектных систем.

  http://www.d6.ru/~sevnhuts/articles/EJBintro.html

- Краткое введение в технологию Enterprise JavaBeans.

  http://www.citforum.ru

- Введение в CORBA.

  http://www.jguru.com

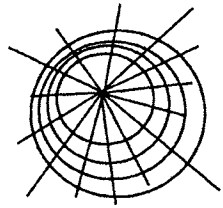
- Ю. Пуха *CORBA/IIOP и Java RMI.* Основные возможности в сравнении.

  http://www.osp.ru

Книги и статьи

- Марина Аншина "Увлекательное путешествие с CORBA 3: по широким просторам распределенных приложений".
- Роберт Орфали, Ден Харки, Джери Эдвардс "Основы CORBA", 1999.
- Сергей Орлик "CORBA3" журнал "Открытые системы", 1999, № 2.
- Douglas C. Schmidt, Steve Vinoski "Introduction to CORBA Messaging" SIGS C++ Report magazine, 1998.
- Douglas C. Schmidt, Steve Vinoski "Portable Object Adapter", SIGS C++ Report magazine.
- Steve Vinoski "New Features for CORBA 3.0".

ГЛАВА 4



Микропрограммы для браузеров. Удаленное и локальное исполнение программ в Интернете/интранете

Способы осуществления различных замыслов и выполнения работ в среде Интернета сегодня столь многообразны, что одно только их перечисление заняло бы не один десяток страниц. Поскольку основным инструментом общения с Сетью остается классический браузер, его же все чаще стремятся использовать в качестве основного рабочего места и с другими информационными системами, построенными в технологии Интернета/интранета (Oracle, IBM WebSphere, Lotus Domino и другими системами ERP, BPR). Посмотрим, что же, все-таки, современный браузер умеет? А умеет он многое. Если раньше классические браузеры типа Netscape и MS Explorer содержали (и сейчас содержат!) встроенную виртуальную Java-машину, то теперь при исполнении Java-микропрограмм в среде браузера эту машину можно запросто обойти и воспользоваться гораздо более новой и мощной, например, из комплекта Sun JDK самых последних версий, а не ждать, когда выйдет новая версия любимого браузера. Помимо исполнения Java-программ, современный браузер умеет и многое другое. Рассмотрим по порядку, как, находясь в среде браузера, можно исполнять любые типы программ и управлять сложными, территориально распределенными системами.

Апплеты

Все начиналось с апплетов. Апплеты были хитами программирования для Интернета в середине 90-х годов. Сейчас их роль снизилась до удобных средств программирования пользовательских интерфейсов и оформления дизайна страниц, тем не менее, включив воображение, современный разработчик, помня об апплетах, изначально обладает богатейшим арсеналом

средств для построения чрезвычайно гибких и безопасных систем любого уровня сложности, т. к. в основе апплетов лежит язык Java.

Виртуальная машина Java

Ядром Java-концепции и системы реализации Java является виртуальная машина Java Virtual Machine (JVM). Это по сути программный микропроцессор с собственным набором инструкций и команд. JVM обеспечивает автоматическое управление памятью, сборку мусора и другие функции для программистов. Имеется несколько вариантов JVM для каждого типа процессора, все они должны быть лицензированы в компании, которой принадлежат исключительные права на эту универсальную технологию — в Sun Microsystems. Исходные коды на C и Java обладают высокой степенью переносимости. Например, IBM использует Java сразу на нескольких платформах: IBM AIX, OS/2, OS/400, OS/390 и Microsoft Windows.

Виртуальная машина Java может исполнять либо апплеты, либо Java-приложения, которые могут взаимодействовать с другими Java-объектами по протоколу HTTP или RMI (рис. 4.1).

Апплет (по замыслу создателей самого этого понятия — сокращение слова application) — это миниатюрная программа на Java (заметим, что размер апплета никто и никогда не ограничивал), которая загружается прямо в браузер и выполняется на машине от его имени, с помощью встроенной в браузер JVM (Java Virtual Machine). Апплет может выполнять любую работу, за исключением права модификации файловых ресурсов (из соображений безопасности это могут делать только *подписанные* апплеты; о технологии цифровых подписей в апплетах будет рассказано ниже). Посмотрим, как браузер выполняет апплет, осуществляющий доступ к СУБД (рис. 4.2).

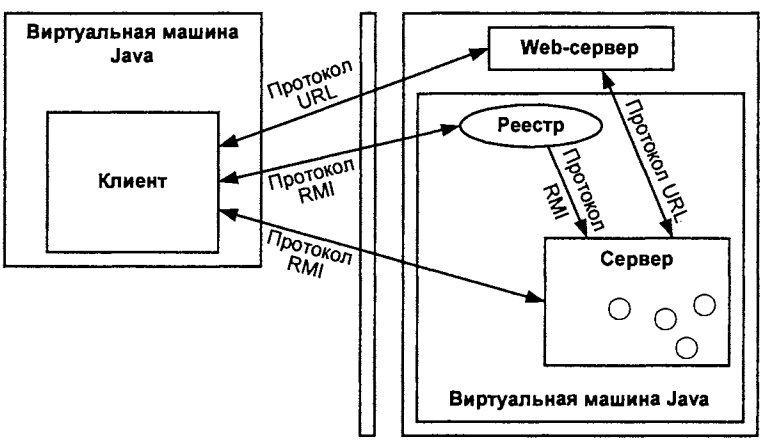


Рис. 4.1. Загрузка и выполнение Java-программ

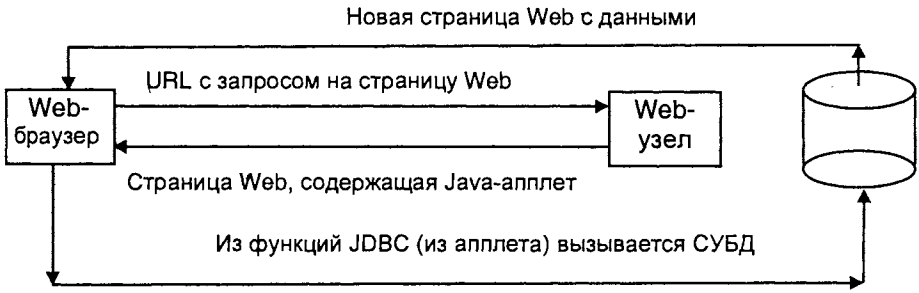


Рис. 4.2. Схема загрузки апплета с доступом к СУБД

Java-апплет — это маленькая прикладная программа, которая может исполняться на любом стационарном, переносном или сетевом компьютере. При загрузке и выполнении апплетов поддерживается клиент-серверная архитектура.

С помощью апплета можно редактировать ввод, управлять экраном и связываться с сервером, который способен обрабатывать посылаемые ему транзакции, например, при взаимодействии с СУБД. Апплеты создаются с использованием специального программного инструментария фирмы SUN JDK (Java Development Kit) или при помощи специальных инструментальных сред, таких как VisualAge from IBM, Symantec's Visual Cafe и Microsoft's Visual J++.

Эти среды содержат визуальные средства программирования и автоматическую генерацию Java-кода (рис. 4.3).

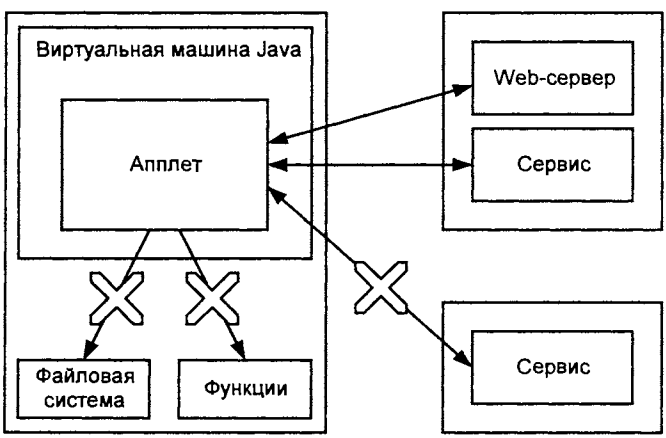


Рис. 4.3. Показывает, как апплет выполняется из Web-браузера

Вызов апплета из HTML-файла также чрезвычайно просто оформляется специальным тегом `<applet>`.

Пример. Файл HelloApplet.html.

```
<applet
   name="HelloApplet"
   code="HelloApplet"
   codebase="file:/e:/sun//src/HelloApplet"
   width="500"
   height="600"
   align="Top"
   alt="Ваш браузер не поддерживает Java.">
</applet>
```

С помощью оператора `<applet>` апплет встраивается в документ. Оператор `<applet>` используется в паре с оператором `</applet>` и имеет следующие параметры:

Параметр	Описание
ALIGN	Выравнивание окна апплета относительно окружающего его текста. Возможны следующие значения: LEFT — выравнивание влево относительно окружающего текста; CENTER — центрирование; RIGHT — выравнивание вправо относительно окружающего текста; TOP — выравнивание по верхней границе; MIDDLE — центрирование по вертикали; BOTTOM — выравнивание по нижней границе
ALT	С помощью этого параметра можно задать текст, который будет отображаться в окне апплета в том случае, если браузер не может работать с апплетами Java
CODE	Имя двоичного файла, содержащего байт-код апплета. Это должно быть имя базового класса, и если он находится в архивном файле типа .JAR, то через пробел с ключевым словом ARCHIVE указывается имя архива (archive=HelloWorld.jar; м. б. несколько архивов, разделенных запятыми). По умолчанию путь к этому файлу указывается относительно каталога с файлом HTML, в который встроен апплет. Такое поведение может быть изменено параметром CODEBASE
CODEBASE	Базовый адрес URL апплета, т. е. путь к каталогу, содержащему апплет в виде URL ссылки
HEIGHT	Ширина окна апплета в пикселах
WIDTH	Высота окна апплета в пикселах
HSPACE	Зазор слева и справа от окна апплета
VSPACE	Зазор сверху и снизу от окна апплета

(окончание)

Параметр	Описание
NAME	Идентификатор апплета, который может быть использован другими апплетами, расположенными в одном и том же документе HTML, а также сценариями JavaScript
TITLE	Строка заголовка

Дополнительно между операторами <applet> и </applet> вы можете задать параметры апплета. Для этого используется оператор <PARAM>.

Основной синтаксис:

<PARAM NAME=имя VALUE=значение>

Например:

```
<PARAM NAME=font VALUE="Courier">
<PARAM NAME=style VALUE="Italic">
<PARAM NAME=size VALUE="15">
<PARAM NAME=bgcolor VALUE="#EEFFFF">
<PARAM NAME=fgcolor VALUE="#0000FF">
```

Параметр CODEBASE задает путь к каталогу локального диска:

```
codebase="file:/e:/sun/src/HelloApplet"
```

Когда вы будете размещать документ HTML на сервере, параметр CODEBASE необходимо изменить или удалить, если этот документ и апплет располагаются в одном каталоге. В любом случае, если параметр CODEBASE задан, он должен указывать адрес URL каталога с апплетом.

Изменение исполняющей Java-машины по умолчанию

По умолчанию исполняющая апплеты виртуальная Java-машина — это JVM (Java Virtual Machine), встроенная в браузер, который запускает HTML-файл с тегом APPLET. Однако, начиная с версии JDK1.2, все это можно изменить и воспользоваться стандартной JDK1.2 фирмы SUN.

Новая версия Sun JDK включает Plug-ins (дополнения), которые, не затрагивая родные JVM браузеров, динамически перенастраивают сами браузеры для работы с JDK1.2, используя тег OBJECT для MS IE 4.0 и тег EMBED для Netscape Navigator. Например, чтобы выполнить апплет в IE4.0, используя в обход Microsoft JVM новую версию SUN JDK1.2, необходимо произвести следующую замену в файле Html:

До замены:

```
......
<A HREF=myproba\proba.java>Source code for proba</A><p>
<APPLET CODE=myproba.proba.class WIDTH=250 HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```

После замены:

```
......
<A HREF=myproba\proba.java>Source code for proba</A><p>
 <OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
    width="600" height="900" align="baseline"
    codebase="http://java.sun.com/products/plugin/1.2/
              jinstall-12-win32.cab#Version=1,2,0,0">
    <PARAM NAME="code" VALUE="myproba.proba.class">

    <PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
            No JDK 1.2 support for APPLET!!</OBJECT>
</APPLET>
</BODY>
</HTML>
```

При этом вы увидите в IE4.0 маленькое приятное окошечко, в котором подтверждается, что работа ведется именно в JDK1.2 (рис. 4.4).

И, соответственно, для Netscape Navigator:

До замены:

```
......
<A HREF=myproba\proba.java>Source code for proba</A><p>
<APPLET CODE=myproba.proba.class WIDTH=250 HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```

После замены:

```
.......
<EMBED type="application/x-java-applet;version=1.2" width="600"
   height="900" align="baseline" code="myproba.proba.class"

   pluginspage="http://java.sun.com/products/plugin/1.2/
                plugin-install.html">
<NOEMBED>
   No JDK 1.2 support for APPLET!!
</NOEMBED>
</EMBED>
```

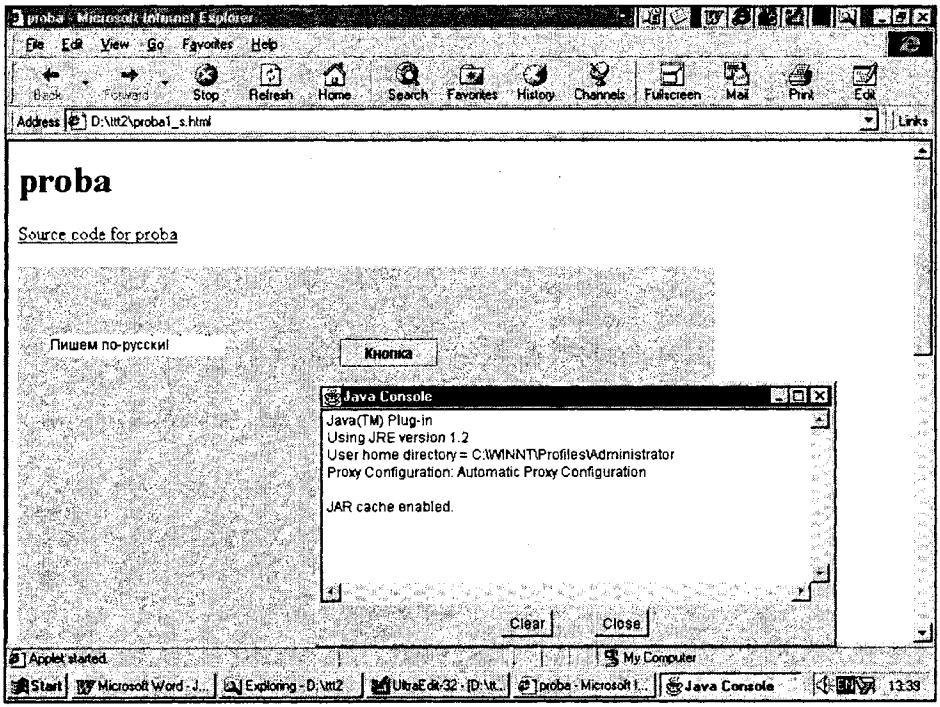


Рис. 4.4. Всплывающее консольное окно Java

Обычно проект апплета в современных условиях создается не вручную, как раньше, а с помощью различных мастеров (Wizards), встроенных в системы программирования, и поэтому, хорошо представляя архитектуру того, что вы пытаетесь создать, вам нужно только правильно отвечать на вопросы, задаваемые мастером.

Вообще, количество разных учебников и документов, описывающих, как проектировать апплеты — просто огромно и мы не будем терять на этом время. Дадим лишь пример наиболее характерного и наиболее типичного апплета, опубликованного в свое время на сайте **http://www.km-cd.com /black_coffee** и принадлежащего Марку Бойнсу (Mark Boyns). При запуске этого апплета маленький симпатичный пингвинчик бегает по узкой полосе на рампе экрана и выстраивает на ней буквы некоего транспаранта. Такой апплет можно использовать в любой программной среде и на любой платформе. Фактически, это наиболее характерный пример проектирования апплета Java.

Пример апплета для работы с буквами

Исходный текст апплета Logo.java

```
import java.applet.*;
import java.awt.*;
```

```
abstract class Thing
{
    Logo parent;
    int x;
    int y;
    int new_x;
    int new_y;
    Image film;
    int frame = 0;
        abstract void paint (Graphics g);
    abstract void erase (Graphics g);
    public void move (int x, int y)
    {
       new_x = x;
       new_y = y;
    }
    public void move (int x)
    {
       new_x = x;
    }

    public int getX ()
    {
       return x;
    }

    public int getY ()
    {
       return y;
    }
}
class Letter extends Thing
{
    Letter (Image film, int frame, int x, int y, Logo parent)
    {
       this.film = film;
       this.frame = frame;
       this.x = x;
       this.y = y;
       new_x = x;
       new_y = y;
    }
    public void paint (Graphics g)
    {
       if (new_x != x || new_y != y)
       {
           erase (g);
```

```
            x = new_x;
            y = new_y;
        }
        Graphics gc = g.create (x, y, 64, 64);
        gc.drawImage (film, 0, -(frame * 64), parent);
        gc.dispose ();
    }

    public void erase (Graphics g)
    {
        g.clearRect (x, y, 64, 64);
    }
}
class Duke extends Thing
{
    boolean walking = true;

    Duke (Image film, int x, int y, Logo parent)
    {
        this.film = film;
        this.x = x;
        this.y = y;
        this.parent = parent;
        new_x = x;
        new_y = y;
    }

    public void walk (int there, int step)
    {
        walking = true;

        if (there > x)
        {
            while (x <= there)
            {
                new_x += step;
                parent.repaint ();
                parent.sleep (100);
            }
        }
        else
        {
            while (x >= there)
            {
                new_x -= step;
                parent.repaint ();
```

```
                parent.sleep (100);
            }
        }
    }

    public void walk ()
    {
        walking = true;
    }

    public void wave (int nwaves)
    {
        walking = false;

        for (int i = 0; i < nwaves; i++)
        {
            parent.repaint ();
            parent.sleep (100);
        }
    }

    public void wave ()
    {
        walking = false;
    }

    public void paint (Graphics g)
    {
        x = new_x;
        y = new_y;

        Graphics gc = g.create (x, y, 55, 64);
        gc.drawImage (film, 0, -((walking ? frame : 2+frame) * 64), parent);
        gc.dispose ();

        frame++;
        if (frame == 2)
        {
            frame = 0;
        }
    }

    public void erase (Graphics g)
    {
        g.clearRect (x, y, 55, 64);
    }
}
```

```
public class Logo extends Applet implements Runnable
{
    int start[] = { -64, -64, 512, 512 };
    int increment[] = { 4, 4, -4, -4 };
    int end[] = { 128, 192, 256, 320 };
    int order[] = { 1, 2, 0, 3 };

    Image letterFilm;
    Letter letters[];
    int currentLetter = 0;

    Image dukeFilm;
    Duke duke;

    Thread thread = null;
    MediaTracker tracker = null;
    Image doubleBuffer;
    boolean paused = false;

    public void init ()
    {
        String param;
        int i;

        tracker = new MediaTracker (this);

        /* Duke */
        dukeFilm = getImage (getCodeBase (), "duke.gif");
        tracker.addImage (dukeFilm, 0);
        duke = new Duke (dukeFilm, -64, 0, this);

        /* Letters */
        try
        {
            param = getParameter ("LETTERS");
            if (param == null)
            {
                param = "sdsu.gif";
            }
        }
        catch (Exception e)
        {
            return;
        }
        letterFilm = getImage (getCodeBase (), param);
        tracker.addImage (letterFilm, 1);
        letters = new Letter[4];
```

```
        for (i = 0; i < 4; i++)
        {
            letters[i] = new Letter (letterFilm, i, start[i], 0, this);
        }

        doubleBuffer = createImage (512, 64);
        resize (512, 64);

        thread = new Thread (this);
        thread.start ();
    }

    public void sleep (long ms)
    {
        try
        {
            Thread.sleep (ms);
        }
        catch (Exception e)
        {
        }
    }

    public void run ()
    {
        int i;

        // Thread.currentThread().setPriority (Thread.MIN_PRIORITY);

        tracker.checkID (0);
        tracker.checkID (1);

        /* Wait for Duke to load. */
        try
        {
            tracker.waitForID (0);
        }
        catch (Exception e)
        {
            return;
        }

        /* Initialize the letters. */
        for (i = 0; i < 4; i++)
        {
            letters[i].move (start[i]);
        }
```

```
duke.move (-64);
duke.walk (230, 20);
duke.wave (13);
duke.walk (-64, 20);

for (i = 0; i < 4;)
{
    repaint ();
    sleep (100);

    currentLetter = order[i];

    /* Проверка, если буква уже размещена. */
    if ((increment[currentLetter] < 0 &&
    letters[currentLetter].getX () <= end[currentLetter])
       || (increment[currentLetter] > 0 &&
       letters[currentLetter].getX () >= end[currentLetter]))
    {
       if (i == 3)
       {
           duke.walk (230, 20);
           duke.wave (25);
           duke.walk (40, 20);
           duke.wave (5);
           duke.walk (-64, 40);
       }
       else if (increment[currentLetter] > 0)
       {
           duke.walk (512, 20);
       }
       else
       {
           duke.walk (-64, 20);
       }

       i++; /* next letter */
    }
    else
    {
       /* Сдвинуть букву. */
       letters[currentLetter].move (letters[currentLetter].
       getX () + increment[currentLetter]);

       /* Привести в движение Duke. */
       if (increment[currentLetter] < 0)
       {
           duke.move (letters[currentLetter].getX () + 40);
       }
```

```
            else
            {
                duke.move (letters[currentLetter].getX () - 40);
            }
        }
    }
}

public void start ()
{
    if (thread == null)
    {
        thread = new Thread (this);
        thread.start ();
    }
}

/* Stop the applet. */
public void stop ()
{
    if (thread != null && thread.isAlive ())
    {
        thread.stop ();
        thread = null;
    }
}

/* Обработчик событий, связанных с мышью. */
public boolean mouseDown (Event e, int x, int y)
{
    if (thread != null)
    {
        thread.stop ();
        thread = null;
    }
    else
    {
        thread = new Thread (this);
        thread.start ();
    }

    return true;
}

/* Обработка событий, связанных с клавиатурой. */
public boolean keyDown (Event e, int key)
{
    switch (key)
```

```
        {
        case ' ':
            if (thread != null)
            {
                if (paused)
                {
                    thread.resume ();
                    paused = false;
                }
                else
                {
                    thread.suspend ();
                    paused = true;
                }
            }
            break;
        }

        return true;
    }

    /* Не очистка экрана, а только перерисовка. */
    public void update (Graphics g)
    {
        paint (g);
    }

    /* Заполняем экран изображением. */
    public void paint (Graphics g)
    {
        Graphics db = doubleBuffer.getGraphics ();
        int i;

        duke.erase (db);
        if (tracker.checkID (1, true))
        {
            for (i = 0; i < 4; i++)
            {
                letters[i].paint (db);
            }
        }
        duke.paint (db);

        g.drawImage (doubleBuffer, 0, 0, this);
        db.dispose ();
    }
}
```

Этот апплет характерен тем, что он использует единственный входной параметр с именем заранее подготовленного графического файла, состоящего из каких-нибудь букв, например java.gif, и встроенную картинку самого Дюка (пингвинчика) (рис. 4.5).

Duke.gif

Java.gif

Рис. 4.5. Заранее подготовленные картинки для апплета

Скомпилируйте этот апплет по команде:

```
javac Logo.java
```

и затем выполните его из файла index.html:

```
<h1 align=center>Пример HTML-файла с встроенным апплетом</h1>
<center>
<table border=4>
<tr><td>
<applet code=Logo.class width=512 height=96>
<param name=letters value=java.gif>
</applet>
</td></tr>
</table>
</center>

<p>
```

Здесь можно посмотреть текст программы этого апплета на Java:

```
<ul>
  <li><a href=Logo.java>Текст программы</a></li>
</ul>
</p>
<p>
```

Необходимые файлы:

```
<ul>
  <li><a href=index.html>Файл запуска апплета</a></li>
  <li><a href=logo.java>Текст программы на Java</a></li>
  <li><a href=duke.gif>Картинка пингвина</a></li>
  <li><a href=java.gif>Картинка из букв</a></li>
</ul>
</p>
```

Наблюдая эту программу в действии, почти любой пользователь, который сможет ее запустить, получит своеобразное удовольствие от своих весьма несложных трудов.

Сервлеты

Java-сервлеты

Если апплеты расширяют функциональность Web-браузеров, то сервлеты расширяют функциональность Web-серверов и в этом их главное отличие. Сервлеты не нужно никуда загружать и они всегда выполняются в контексте Web-сервера, обеспечивая, в отличие от обычных CGI-процессов или скриптов, куда более развитые возможности. Когда-то казалось, что основное русло программирования для Интернета будет проложено именно сервлетами, но это оказалось не так. *Сервлеты* являются объектами, которые образуют специфический интерфейс, с помощью которого их можно свободно встраивать в Java-ориентированные Web-серверы. В отличие от апплетов, они являются облегченными Java-объектами (без графики и GUI-компонентов), но зато, будучи кроссплатформенными и динамически загружаемыми и имея в своем распоряжении все возможности Java, они могут использовать практически любой HTML и взаимодействовать с любым браузером. Именно это, как ни странно, — т. е. их принудительное HTML-взаимодействие с браузером, — сейчас ощущается как их недостаток. Сервлеты лишены главных преимуществ апплетов — графического программирования пользовательских интерфейсов, поэтому в настоящее время связки типа "апплет-сервлет", которые могли бы дополнять друг друга, практически никем не используются из-за обремененности сервлетов http-классами, которые апплетам совсем не нужны. Куда проще и эффективнее оказалось запускать программную пару "апплет-RMI-сервер" (или CORBA-сервер) и апплет из браузера. Именно по этому принципу выполняются ныне все административные функции (Administration Manager) для СУБД. Но в этом случае браузер выполняет чисто декоративную роль и не участвует в работе. Тем не менее для тех программистов, которые обожают язык HTML, сервлеты представляются чрезвычайно удобным средством.

Сервлеты обслуживают Web-процесс, используя диалог в качестве парадигмы.

Если рассуждать обобщенно, то потоковый процесс с использованием сервлетов выглядит следующим образом:

- клиент посылает запрос к серверу;
- сервер переадресует запрос к сервлету;
- сервлет строит ответ и передает его серверу. Содержимое ответа создается динамически и зависит от запроса клиента.

Рис. 4.6 иллюстрирует этот процесс.

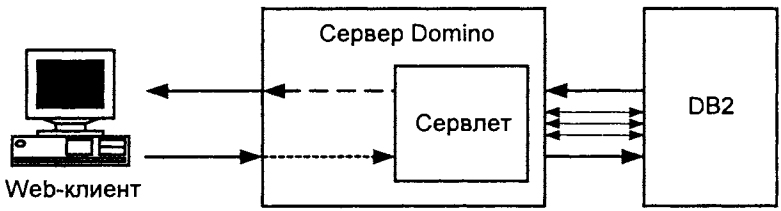


Рис. 4.6. Процесс прохождения запроса

С точки зрения программиста сервлет — это обычная программа. Сервлет импортирует стандартные Java классы и использует Java servlet API. Сервлеты могут загружаться, когда Web-сервер стартует, и резидентно оставаться в памяти, дожидаясь, когда их вызовут. Поскольку сервлеты являются готовыми для исполнения объектами в форме байт-кодов, они могут загружаться и динамически. Так как сервлеты никогда не применяют графический интерфейс, иногда их еще называют безликими объектами.

Кроме того, сервлеты используют для доступа к базам данных, программируя все удаленные операции с базой данных из браузера. В настоящее время большинство Web-серверов поддерживают технологию сервлетов, но лучше всего показать их конструкцию на базе Java Web Server. Этот сервер хорош тем, что он наиболее полно реализует технологию сервлетов. Стандартный Servlet API реализован как в самом Java Web Server, так и в отдельном продукте JSDK, но, начиная с версии 1.2, он включен в состав JDK. Характерной особенностью сервлетов является то, что они не требуют создания новых процессов при каждом новом запросе. Множество сервлетов выполняются параллельно в рамках одного процесса на сервере и по своей производительности превосходят как приложения CGI, так и Fast-CGI (рис. 4.7).

Несмотря на то, что сервлеты используют HTTP-протокол, им нет необходимости перезагружать процесс при каждом новом запросе, и это также повышает их быстродействие. Каким образом программируются операции с базой данных на сервере? Сервлет не накладывает никаких ограничений на использование интерфейсов с базой данных. Это может быть обыкновенный JDBC-ODBC или специфические драйверы, ориентированные на конкрет-

ные SQL-серверы и базы данных. Рассмотрим типичный DBServlet.java, который с успехом будет выполняться как на Java Web Server, так и Web-сервере Apache.

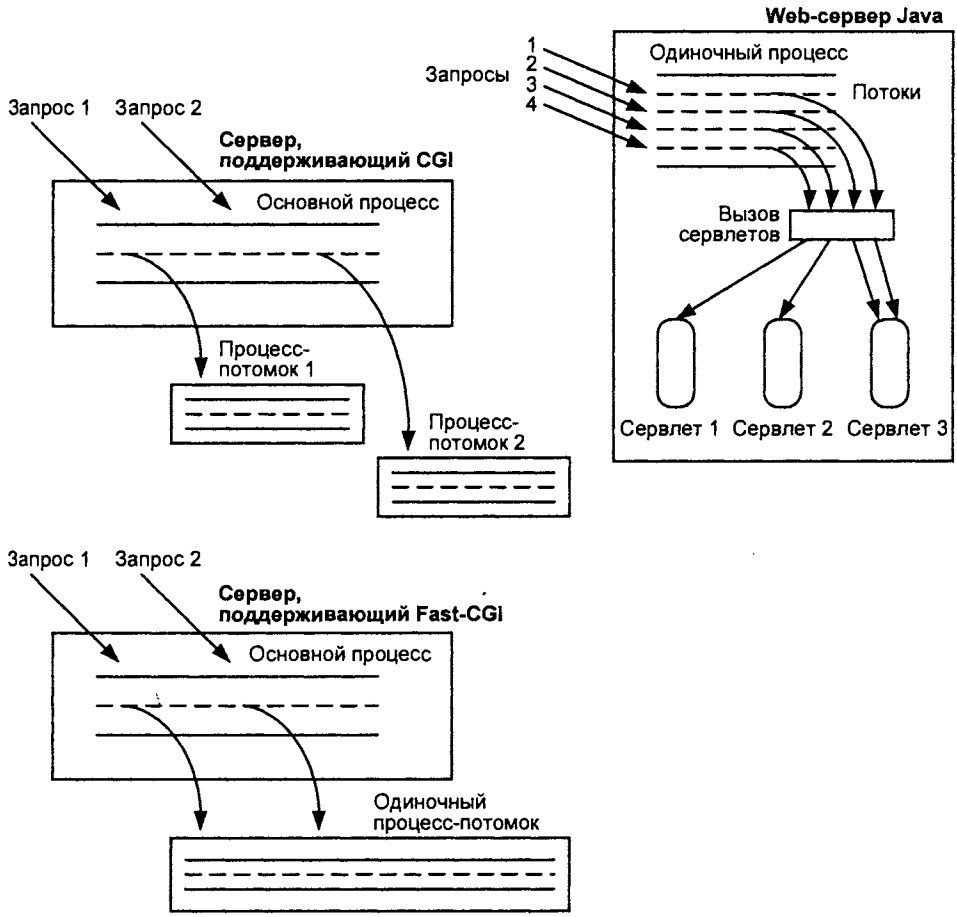


Рис. 4.7. Возможные процессы на Web-сервере

```
/*
 * @(#)DBServlet.java 1.6 97/06/13
 */
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.net.*;
import java.sql.*;
/**
```

```
 * DBServlet
 *
 * Этот сервлет демонстрирует, как использовать JDBC-стек
 * и получать доступ к базам данных при помощи сервлетов.
 * Для того чтобы выполнить этот сервлет,
 * переменные CLASSPATH, LD_LIBRARY_PATH, ODBCINI
 * должны быть соответствующим образом установлены.
  */

public class DBServlet extends HttpServlet {

    /**
     * init method
     */
    public void init(ServletConfig conf) throws ServletException {
        super.init(conf);
    }

    /**
     * service method
     */
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {

        String stack, username, password, query;
       Connection con;
       PrintStream out;
/*     ServletOutputStream out = res.getOutputStream();*/

       // Загрузка параметров. Именно эти параметры должны будут
       // вводиться в специально прорисованном окне браузера, с которым
       // будет взаимодействовать наш сервлет

       stack = req.getParameter("stack");
       username = req.getParameter("username");
       password = req.getParameter("password");
       query = req.getParameter("query");

       res.setContentType("text/html");
       out = new PrintStream(res.getOutputStream());
       printPageHeader(out);

       // в том случае, если параметры отсутствуют
       if (stack == null || username == null || password == null ||
           query == null) {
```

```
            printPageFooter(out);
            return;
        }

        String url   = "jdbc:odbc:"+stack;

        out.println("<hr><h3>Previous Query</h3>");
        out.println("<pre>");
        out.println("Database Stack : "+stack);
        out.println("       User ID : "+username);
        out.println("      Password : "+password);
        out.println("         Query : "+query);
        out.println("</pre>");

        try {
            // Найти jdbc-стек. Может существовать несколько
            // зарегистрированных драйверов
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            // Class.forName("jdbc.odbc.JdbcOdbcDriver");

            // Получить соединение с базой данных
            con = DriverManager.getConnection(url, username, password);
            out.println("<hr>");
            out.println("<h3>ODBC Driver and Database Messages</h3>");
            checkForWarning(con.getWarnings(), out);
            DatabaseMetaData dma = con.getMetaData();
            out.println("Connected to " + dma.getURL() + "<br>");
            out.println("Driver       " + dma.getDriverName() + "<br>");
            out.println("Version      " + dma.getDriverVersion() + "<br>");

            // Создать и выполнить запрос. Конкретный оператор SQL
            // вводится удаленным пользователем в окне его браузера
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);

// Печать результатов. Они переназначаются на стандартный вывод
// и поступают в браузер клиента
            dispResultSet(rs, out);
            rs.close();
            stmt.close();
            con.close();
            out.println("<hr>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
              out.println("SQLState: " + ex.getSQLState() + "<br>");
              out.println("Message:  " + ex.getMessage() + "<br>");
```

```
            out.println("Vendor:   " + ex.getErrorCode() + "<br>");
            ex = ex.getNextException();
         }
      }
      catch (java.lang.Exception ex) {
          ex.printStackTrace();
      }

      printPageFooter(out);
   }

   /**
    * Возврат запрошенной информации
    */
   public String getServletInfo() {
return "Это простой сервлет, который показывает, как использовать JDBC";
   }


   /*
    * Проверка и печать информации сервера
    */
   private void checkForWarning(SQLWarning warn, PrintStream out)
         throws SQLException
   {
      boolean rc = false;

      if (warn != null) {
         out.println("<hr>*** Warning ***<p>");
         rc = true;
         while (warn != null) {
            out.println("SQLState: " + warn.getSQLState() + "<br>");
            out.println("Message:  " + warn.getMessage() + "<br>");
            out.println("Vendor:   " + warn.getErrorCode() + "<br>");
            warn = warn.getNextWarning();
         }
      }
   }


   /*
    * Показ результатов запроса в табличном html-формате
    */
   private void dispResultSet(ResultSet rs, PrintStream out)
      throws SQLException, IOException

   {
      int i;
```

```
    // Метаданные используются для получения  информации о схеме
    ResultSetMetaData rsmd = rs.getMetaData();
    int numCols = rsmd.getColumnCount();
    out.println("<hr>");
    out.println("<h3>Database Columns and Data</h3>");
    out.println("<table border=3>");
    out.println("<tr>");
    for (i=1; i<=numCols; i++) {
       out.println("<th>" + rsmd.getColumnLabel(i) + "</th>");
    }
    out.println("</tr>");

    // Для всех данных
    while (rs.next()) {
      out.println("<tr>");

      // for one row
       for (i=1; i<=numCols; i++) {
         dispElement(rs, rsmd.getColumnType(i), out, i);
       }
      out.println("</tr>");
    }
    out.println("</table>");
}


// Печать одного элемента
private void dispElement(ResultSet rs, int dataType,
                         PrintStream out, int col)
      throws SQLException, IOException

{
     String cp1 = new String("Cp1251");
 // В зависимости от типа данных, определяем различные типы обработки
    switch(dataType) {
    case Types.DATE:
      java.sql.Date date = rs.getDate(col);
      out.println("<th>" + date.toString() + "</th>");
      break;
    case Types.TIME:
      java.sql.Time time = rs.getTime(col);
      out.println("<th>" + time.toString() + "</th>");
      break;
    case Types.TIMESTAMP:
      java.sql.Timestamp timestamp = rs.getTimestamp(col);
```

```
          out.println("<th>" + timestamp.toString() + "</th>");
          break;
        case Types.CHAR:
        case Types.VARCHAR:
        case Types.LONGVARCHAR:
          String str = rs.getString(col);
// Возможно, что здесь вам понадобятся кириллические преобразования
          out.println("<th>" + str + "</th>");
          break;
        case Types.NUMERIC:
        case Types.DECIMAL:
          java.math.BigDecimal numeric = rs.getBigDecimal(col, 10);
          out.println("<th>" + numeric.toString() + "</th>");
          break;
        case Types.BIT:
          boolean bit = rs.getBoolean(col);
          out.println("<th>" + new Boolean(bit) + "</th>");
          break;
        case Types.TINYINT:
          byte tinyint = rs.getByte(col);
          out.println("<th>" + new Integer(tinyint) + "</th>");
          break;
        case Types.SMALLINT:
          short smallint = rs.getShort(col);
          out.println("<th>" + new Integer(smallint) + "</th>");
          break;
        case Types.INTEGER:
          int integer = rs.getInt(col);
          out.println("<th>" + new Integer(integer) + "</th>");
          break;
        case Types.BIGINT:
          long bigint = rs.getLong(col);
          out.println("<th>" + new Long(bigint) + "</th>");
          break;
        case Types.REAL:
          float real = rs.getFloat(col);
          out.println("<th>" + new Float(real) + "</th>");
          break;
        case Types.FLOAT:
        case Types.DOUBLE:
          double longreal = rs.getDouble(col);
          out.println("<th>" + new Double(longreal) + "</th>");
          break;
        case Types.BINARY:
```

```
        case Types.VARBINARY:
        case Types.LONGVARBINARY:
          byte[] binary = rs.getBytes(col);
          out.println("<th>" + new String(binary, 0) + "</th>");
          break;
        }
    }


    private void printPageHeader(PrintStream out) {
        out.println("<html>");
        out.println("<head>");
        out.println("<tltle>Типичный сервлет для работы с БД</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<center><font size=5>" +
                    "<b>Jeeves Database Servlet</b>" +
                    "</font></center>");
        out.println("<hr>");
        out.println("<form action=\"/servlet/dbServlet\"
method=\"get\">");
        out.println("<pre>");
        out.println("ODBC  DSN : <input type=textarea name=stack>");
        out.println("User ID : <input type=textarea name=username>");
        out.println("Password : <input type=textarea name=password>");
        out.println(" SQL   Query : <input type=textarea name=query>");
        out.println("</pre>");
        out.println("<input type=submit>");
        out.println("</form>");
    }


    private void printPageFooter(PrintStream out) {
         out.println("</body>");
        out.println("</html>");
        out.flush();
    }
}
```

В этой программе содержатся пакеты классов для поддержки сервлетов — javax.servlet.* и javax.servlet.http.*, которые находятся в специальном продукте фирмы JavaSoft JSDK1.0.1 или непосредственно в JDK1.2.

Главная вещь, которую необходимо при этом понять, — это каким образом извлекаются аргументы, переданные сервлету на вводе, и каким образом передаются ответные параметры на выводе.

```
public void service (HttpServletRequest req,
                     HttpServletResponse res) throws
                     ServletException, IOException
    {
        ...
    }
```

Специальный метод `request` содержит список параметров, которые могут быть извлечены при помощи метода `HttpServletRequest::getParameterNames`.

```
stack = req.getParameter("stack");
```

возвращает значение определенного вами в html-форме параметра `stack`, в котором вы можете задавать с клавиатуры имя ODBC-источника (ODBC DSN).

Можно установить специальный цикл обработки для извлечения каждого параметра, переданного из HTML-формы, когда она направлена для обработки к сервлету:

```
while ( values.hasMoreElements())
            {
                ...
            }
```

Имя и значение каждого извлекаемого параметра можно получать во время работы цикла.

```
String name, value;
                name = (String)values.nextElement();
                value = req.getParameter (name);
```

Этих знаний вполне достаточно для того, чтобы начать писать простые сервлеты, расширяющие функциональность JavaServer для обработки форм, вместо того, чтобы писать сложные CGI. Имеется, впрочем, еще одна проблема, которую также необходимо изучить, — это обработка событий и исключительных ситуаций. Для этого необходимо специфицировать соответствующий сервис и метод `init` в подклассе `HttpServlet` как `ServletException`:

```
    public void init (ServletConfig conf) throws ServletException
    ...
    public void service (HttpServletRequest req,
                         HttpServletResponse res) throws
                         ServletException, IOException
    ...
```

Внутри сервлета DBServlet в методе `service` можно использовать один или несколько блоков `try/catch`, которые обеспечивают пользователю или про-

граммисту обработку событий, связанных с некоторыми неверными действиями или ситуациями. В первом блоке `try` пользователь может обработать ситуацию при возникновении проблемы соединения с сервером СУБД или при выполнении sql-запроса. Второй блок `try` можно использовать по своему усмотрению, например, при записи результатов в файл и последующей проверке — должным ли образом он закрыт перед выходом из обслуживающей процедуры. Если нет — в стандартное устройство вывода выдается соответствующая трассировка.

Эту программу легко модифицировать в соответствии с вашими потребностями и получать не только содержимое таблиц и колонок, но и любые метаданные, потому что в стандартном пакете java.sql содержится гораздо больше методов, по сравнению с теми, что предлагают другие стандартные средства для построения ODBC-приложений.

По поводу сервлетов возникает очень много вопросов в связи с корректным преобразованием кириллических символов. К сожалению, как и всегда, по этой теме, на сегодня не существует единого универсального решения, и поэтому методы кириллизации сервлетов могут меняться с каждой новой реализацией JDK или JSDK. На мой взгляд, наиболее квалифицированными экспертами в "русском вопросе" для Java на сегодня являются Сергей Астахов, Сергей Безруков, Алексей Менделев и Вячеслав Педак. К их статьям в Интернете я и отсылаю любознательных читателей.

http://www.comita.spb.ru/users/sergeya/java/ruschars.html

Что же касается сервлетов, то скажу только, что когда сервлет посылает ответ клиенту, то он может сделать это разными способами: с помощью метода `OutputStream`, метода `PrintWriter` или как-то еще. Если используется метод `OutputStream`, можно задействовать любые методы перекодировки символов, потому что в этом случае записываются массивы байтов. Если вы предпочитаете метод `PrintWriter`, то неплохо бы сначала позаботиться об установлении необходимой кодировки через вызов:

```
response.setContentType("text/html; charset=windows-1251"));
```

Когда сервлет обрабатывает входные параметры, присылаемые ему из браузера, следует помнить о том, что браузеры кодируют русские буквы, используя установленную на машине пользователя языковую кодировку по умолчанию и ничего о ней сервлету не сообщают. Что из этого следует? Из этого следует весьма неприятный факт, что когда при раскодировке символов сервлет применяет методы `HttpUtils.parsePostData()` и `HttpUtils.parseQueryString()`, то они просто обнуляют старший байт и вы никогда не узнаете о том, с какой кодировкой в данный момент необходимо иметь дело. Что делать? Можно добавить специальное поле в форму, но захочет ли неквалифицированный пользователь его заполнять?

Сервлеты на базе Apache

Для тех, кто имеет дело с различными компьютерными платформами и стремится к универсализму, выбор Apache HTTP-сервера является, пожалуй, самым подходящим. Разве плохо, когда написанные вами модули могут работать и на ПК, и на многопроцессорных высокопроизводительных системах, причем инсталляции и компоновки рабочих программ, как правило, не требуется, дело сводится к простому копированию. Можно не выходя из офиса перемещать рабочие программы в специальные каталоги Web-сервера, находящегося за тридевять земель, и они мгновенно могут быть востребованы десятками пользователей, которым нет необходимости загружать их на свои компьютеры. До недавнего времени все Web-серверы могли вызывать на выполнение только три типа процессов: CGI, ISAPI и NSAPI, причем два последних чаще всего были реализованы на Wintel-платформе. Так, с появлением Delphi-3 многие стали предлагать свои разработки в виде Web-модулей, но аппетиты пользователей умерялись разработчиками, которые могли предложить только вариант Web, работающий под Windows-95/NT, и для связи с СУБД на больших машинах приходилось применять некое подобие Gateway-ODBC. Такой вариант не всегда оказывался эффективным, тем более, что протокол HTTP при каждом новом запросе требовал перезагрузки модуля. Применение сервлетов в связке с СУБД решает эту проблему. Для Apache существует специальный программный компонент JServ, который так же, как и сам сервер в исходных кодах, распространяется совершенно бесплатно. При запуске Apache автоматически стартует Java-обработчик, которому и будут передаваться на исполнение все запросы, связанные с запуском Java-программ. Наоборот, весь вывод из Java переадресуется непосредственно в Web, который аккуратным образом передает его запросившему браузеру. Скорость выполнения Java-приложений зависит от производительности компьютера, на котором сконфигурирован Web-сервер Apache. Такой подход чрезвычайно гибок, ибо позволяет постоянно обновлять Java-среду, наращивать библиотеку классов и ничего не изменять в настройках Apache. Впрочем, и в самих конфигурационных файлах Apache для программистов имеется масса полезных вещей, они свободно корректируются, открыты для добавлений и предоставляют гораздо больше простора для маневра по сравнению с жестко запрограммированными системами для администрирования Web-узлов, такими как Fastrack и Java Web Server. Для того чтобы использовать Apache во взаимодействии с Java, нужно инсталлировать продукт Jserv, содержащий необходимые библиотеки Java-классов, затем изменить конфигурационный файл Apache Configuration, добавив в него строку:

```
Module jserv_module  mod_jserv.o
```

(Модуль mod_jserv.c должен находиться при этом в каталоге Apache src/.) После этого необходимо выполнить повторную компиляцию Apache с по-

мощью программы make. Затем можно изменить основной конфигурационный файл Apache httpd.conf, добавив в него строки вида:

```
<IfModule mod_jserv.c>
  ServletProperties     conf/servlet.properties
  ServletPort           8007
  ServletBinary         /opt1/jdk1.1.5/bin/java
  ServletClassPath      /opt1/jdk1.1.5/lib/classes.zip
  ServletClassPath      /opt1/local/etc/httpd/jserv/lib/servclasses.zip
  ServletClassPath      /opt1/local/etc/httpd/jserv/classes
  ServletClassPath      /opt1/JSDK1.0.1/lib/classes.zip
  ServletErrorLog    logs/jserv_log
  ServletAlias         /servlets /opt1/local/etc/httpd/jserv/servlets
  ServletAuthExport     Off
#      AddHandler     jhtml-parser .jhtml
#      Action         jhtml-parser /servlets/org.apache.jserv.JServSSI
</IfModule>
```

Разумеется, в каждом конкретном случае каталоги могут быть разными, однако, сразу после старта Apache HTTP-сервера по команде `httpd`, он автоматически запускает Java-обработчик, который и будет загружать и исполнять все запрошенные сервлеты, в том числе и те, что будут работать с базами данных. Преимущество Apache при работе с базами данных заключается в том, что, поскольку он так же, как и Java, доступен в исходных кодах, можно эффективно использовать его расширяемую архитектуру и дополнять различными промежуточными фильтрами на Java и на языке C, которые по усмотрению разработчиков могут выполнять различные полезные функции.

Аглеты

Что такое аглет и Java Aglet API (J-AAPI)

Аглет (или мобильный агент) — это Java-объект, который может перемещаться от одного Интернет-хоста к другому. В отличие от обычных Java-объектов, аглет может внезапно приостановить свое выполнение на одном хосте, переместиться на другой хост и там продолжить свое выполнение. Когда аглет перемещается, он переносит на удаленный хост не только свой программный код и свое состояние, но и состояние данных.

Java Aglet Application Programming Interface (J-AAPI) — это стандартный интерфейс для программирования аглетов. J-AAPI определяет различные методы для:

- создания аглетов;
- управления сообщениями;
- инициализации;

- диспетчеризации;
- ретракции;
- активации/дезактивации;
- клонирования;
- диспозиции и перемещения.

Главные цели создания

Не совсем понятно, для чего создавалась эта технология, помимо демонстрации уникальных возможностей Java. Марк Гиббс в своей известной статье в журнале "Сети" определяет аглет как "механического слугу, который выполняет любой приказ и освобождает человека для более творческой работы". На самом деле не все так просто, и не случайно, что аглеты не нашли пока повсеместного применения — примеров их использования чрезвычайно мало. По сути, для нас — это всего лишь экзотика. Конечно, можно было бы предположить, что агент сам должен решить, где ему наиболее выгодно выполняться: отправился бы за поиском данных на сервер СУБД, там бы что-нибудь нашел и вернулся на исходную машину с результатами поиска. В принципе, все возможно. Для этого надо скачать с узла SUN Aglets SDK (пакет это совсем небольшой, 2—3 Мбайт), изучить его и написать, используя многочисленные примеры, содержащиеся в пакете, собственное приложение. Но зачем? Стоит ли шкурка выделки? Архитектура аглетов довольна сложна и не так просто с ходу в ней разобраться.

Архитектура аглетов

Передача сообщений между аглетами или между аглетом и пользователем выполняется через специальных посредников (proxy) (рис. 4.8).

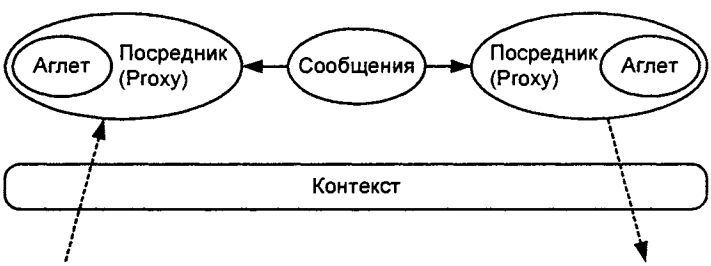


Рис. 4.8. Использование посредников

Посредник скрывает положение аглета, не допуская прямого доступа к его коду и доступа со стороны других аглетов. Аглеты пересылаются из одного контекста в другой по протоколу ATP (Agent Transfer Protocol), разработан-

ному IBM. Этот протокол уровня приложения не зависит от платформы и использует URL для определения местоположения агентов и серверов. Впрочем, при программировании аглетов могут использоваться и другие протоколы (рис 4.9).

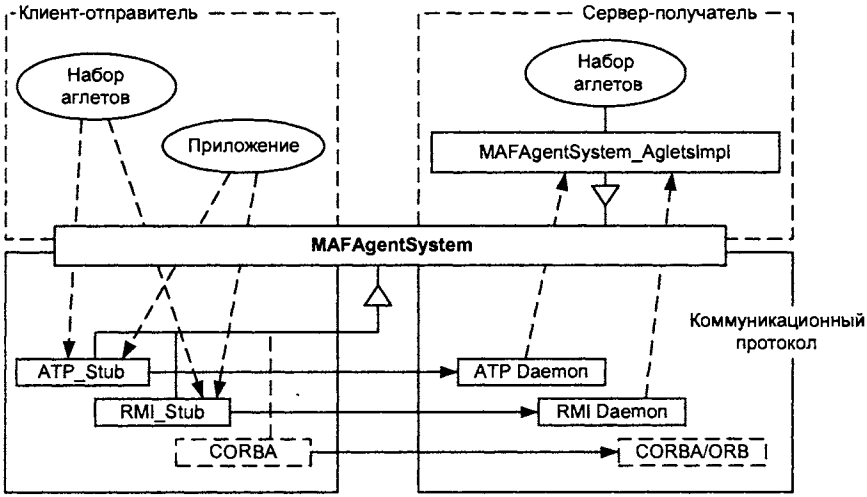


Рис. 4.9. Архитектурная модель Аглет-технологии

Существует такое понятие, как аглет-объект и его жизненный цикл (рис. 4.10). Класс `Com.ibm.aglet.Aglet` обеспечивает базовый набор для мобильных агентов и каждый аглет должен быть экземпляром его подклассов. Чтобы использовать аглет, вы должны создать его экземпляр (инстанцию). Первое, что нужно сделать — это вызвать `AgletContext.createAglet(URL codebase, String name, Object init)`. Данный примитив создает новую инстанцию внутри специфицированного контекста и производит начальную инициализацию, а затем вызывает метод `Aglet.onCreation(Object init)`, в результате которого создается аглет-объект. Для создания копии существующего аглета можно использовать примитив `Aglet.clone()`. Клонированный аглет имеет то же состояние, что и исходный объект, но другой ID. После этого аглет с помощью специальных методов может перемещаться на удаленные узлы, возвращаться на исходную машину и выполнять свою работу.

Когда аглет заканчивает работу в контексте, он может переместиться в другой контекст или в контекст, в котором был создан. Аглеты также можно принудительно возвращать в контекст: переслав аглет, вы оставляете его накапливать данные до тех пор, пока вам не понадобится, чтобы он вернул результаты.

В заключение данного раздела, приведем пример простейшего аглета из состава стандартного дистрибутива Aglets SDK Sun Microsystems (ASDK 1.1 beta).

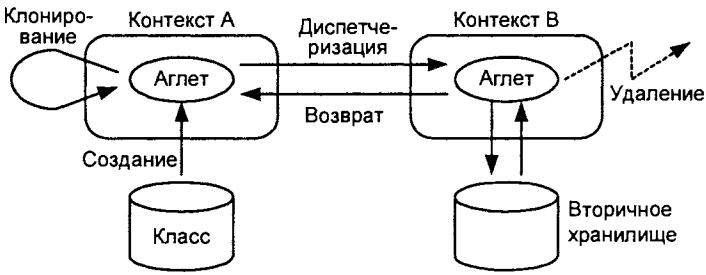


Рис. 4.10. Аглет-объект и его жизненный цикл

```
package examples.hello;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import com.ibm.aglet.util.*;

import com.ibm.agletx.util.SimpleItinerary;

import java.lang.InterruptedException;
import java.io.Externalizable;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.io.IOException;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;

/**
 * <tt> HelloAglet </tt> 'это мобильный агент, который самостоятельно
 * перемещается на удаленный хост, чтобы сказать "Hello World",
 * а затем возвращается обратно и "помирает".
 * <p>1. Показывает диалоговое окно для ввода имени удаленного сервера.
 * вызов метода handleMessage().
 * <p>2. (Дать сообщение "startTrip") Уйти на удаленный сервер,
 * чтобы оттуда послать сообщение "sayHello"
 * самому себе.
 * <p>3. (Дать сообщение "sayHello") Показать строку в течение 5 секунд
 * и вернуться назад. Послать сообщение "atHome" самому себе.
 * <p>4. (Дать сообщение "atHome") Распечатать строку "Я вернулся"
 * в течение 2 секунд и разрушить самого себя.
 *
  * @author      Danny B. Lange
 * @author      Mitsuru Oshima
 */
```

```
public class HelloAglet extends Aglet {

    /*
     * UI to interact with a user
     * this will be automatically disposed when the aglet is disposed
     */
    transient Frame my_dialog = null;  // not serialized

    /*
     * message
     */
    String message = null;

    /*
     * home address represented as a string
     */
    String home = null;

    /*
     * Itinerary
     */
    SimpleItinerary itinerary = null;

    /*
     * Initializes the aglet.
     * Only called the very first time this aglet is created.
     */
    public void onCreation(Object init) {
        setMessage("Hello World!");  //default message

        // Create GUI to control this Aglet
        createGUI();
        itinerary = new SimpleItinerary(this);

        // Initialize the variable.
        home = getAgletContext().getHostingURL().toString();
    }

    protected void createGUI(){
        my_dialog = new MyDialog(this);

        my_dialog.pack();
        my_dialog.setSize(my_dialog.getPreferredSize());
        my_dialog.setVisible(true);
    }

    /*
    * Set the message
```

```
    * @param message the message to send
    */
    public void setMessage(String message){
        this.message = message;
    }

    /*
     * Handles the message
     * @param msg the message sent
     */
    public boolean handleMessage(Message msg) {
        if (msg.sameKind("atHome")) {
            atHome(msg);
        } else if (msg.sameKind("startTrip")) {
            startTrip(msg);
        } else if (msg.sameKind("sayHello")) {
            sayHello(msg);
        } else if (msg.sameKind("dialog")) {
            dialog(msg);
        } else {
            return false;
        }
        return true;
    }

    /*
     * Reports arrival home and disappears
     */
    public void atHome(Message msg) {
        setText("I'm back.");  // greetings
        waitMessage(2 * 1000); // show message, 2 seconds
        dispose();             // dispose it self
    }

    /**
     * Strats the trip of this aglet to the destination.
     */
    public synchronized void startTrip(Message msg) {
        // get the address for trip
        String destination = (String)msg.getArg();

        // Go to the destination and Send "sayHello" message to
owner(this)
        try {
            itinerary.go(destination, "sayHello");
        } catch (Exception ex) {
```

```
            ex.printStackTrace();
        }
    }

    /*
     * Say hello!
     */
    public void sayHello(Message msg) {
        setText(message);          // greetings
        waitMessage(5 * 1000);     // show message, 5 seconds

        // try back home
        try {
            setText("I'll go back to.. " + home);
            waitMessage(1000);  // 1 second
            // Go back home and Send "atHome" message to owner this
            itinerary.go(home, "atHome");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * Creates and shows the dialog window.
     * This aglet keeps the reference to the instance
     * of the Dialog to avoid opening multiple dialog windows.
     */
    public void dialog(Message msg) {
        // check and create a dialog box
        if (my_dialog == null) {
            my_dialog = new MyDialog(this);
            my_dialog.pack();
            my_dialog.setSize(my_dialog.getPreferredSize());
        }

        // show the dialog box
        my_dialog.setVisible(true);
    }
}


/*
 * MyDialog class is the window to be opened when the dialog required.
 * This is NOT a subclass of Dialog.
 */
class MyDialog extends Frame implements WindowListener, ActionListener{
```

```
/*
 * The aglet a user interacts with.
 */
private HelloAglet aglet  = null;

/*
 * UI Components
 */
private AddressChooser dest = new AddressChooser();
private TextField msg       = new TextField(18);
private Button go           = new Button("GO!");
private Button send         = new Button("Send CLONE!");
private Button close        = new Button("CLOSE");

/*
 * Constructs the dialog window
 * @param aglet The aglet the user interacts with.
 */
MyDialog(HelloAglet aglet) {
    this.aglet = aglet;
    layoutComponents();

    addWindowListener(this);
    go.addActionListener(this);
    send.addActionListener(this);
    close.addActionListener(this);
}

/*
 * Layouts all components
 */
private void layoutComponents() {
    msg.setText(aglet.message);

    // Layouts components
    GridBagLayout grid = new GridBagLayout();
    GridBagConstraints cns = new GridBagConstraints();
    setLayout(grid);

    cns.weightx = 0.5;
    cns.ipadx = cns.ipady = 5;
    cns.fill = GridBagConstraints.HORIZONTAL;
    cns.insets = new Insets(5,5,5,5);

    cns.weightx = 1.0;
    cns.gridwidth = GridBagConstraints.REMAINDER;
    grid.setConstraints(dest, cns);
    add(dest);
```

```
    cns.gridwidth = GridBagConstraints.REMAINDER;
    cns.fill = GridBagConstraints.BOTH;
    cns.weightx = 1.0;
    cns.weighty = 1.0;
    cns.gridheight = 2;
    grid.setConstraints(msg, cns);
    add(msg);

    cns.weighty = 0.0;
    cns.fill = GridBagConstraints.NONE;
    cns.gridheight = 1;

    Panel p = new Panel();

    grid.setConstraints(p, cns);
    add(p);
    p.setLayout(new FlowLayout());
    p.add(go);
    p.add(send);
    p.add(close);
}


/**
 * Handles the action event
 * @param ae the event to be handled
 */
public void actionPerformed(ActionEvent ae){
    // execute "GO!" command
    if("GO!".equals(ae.getActionCommand())){
        aglet.setMessage(msg.getText());
        try{
            AgletProxy p = aglet.getProxy();
            p.sendOnewayMessage(new Message("startTrip",
                                            dest.getAddress()));
        }catch(Exception e) {
            e.printStackTrace();
        }
    }

    // execute "Send CLONE!" command
    else if("Send CLONE!".equals(ae.getActionCommand())){
        aglet.message = msg.getText();
        try {
             AgletProxy p = (AgletProxy)aglet.clone();
             p.sendOnewayMessage(new Message("startTrip",
             dest.getAddress()));
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        // execute "CLOSE" command
        else if("CLOSE".equals(ae.getActionCommand())){
            setVisible(false);
        }
    }

    /**
     * Handles the window event
     * @param we the event to be handled
     */

     public void windowClosing(WindowEvent we){
        dispose();
     }

     public void windowOpened(WindowEvent we){
     }

     public void windowIconified(WindowEvent we){
     }

     public void windowDeiconified(WindowEvent we){
     }

     public void windowClosed(WindowEvent we){
     }

     public void windowActivated(WindowEvent we){
     }

     public void windowDeactivated(WindowEvent we){
     }
}
```

Естественно, чтобы выполнить данный пример, нужно правильно установить все переменные, как это описано в ASDK, запустить на обеих сторонах аглет-сервера демонов (на Unix — программа agletsd, которая запускается по командной строке: `% $AGLET_HOME/bin/agletsd -f my_aglets.props`).

Не следует думать, что на удаленном хосте никто и понятия не имеет, что к ним пожалует некий аглет, — нет, все вопросы администрирования и защиты, как и всегда в Java, продуманы очень тщательно, но это уже вопрос чистой техники. Главное — это понять базовые принципы функционирования аглетов.

Скриптлеты

Сначала немного о скриптах

Подобно языку JavaScript, впервые изобретенному Netscape и полностью продублированным Microsoft под именем JScript в MS Internet Explorer 3.0, Visual Basic Script (VBScript) дает возможность внедрения команд процедурного языка в HTML-документ. Так же, как и JavaScript, VBScript — это интерпретируемый язык! MS Internet Explorer интерпретирует команды бэйсика и выполняет их.

VB-сценарий является быстрым и гибким подмножеством Visual Basic и Visual Basic Application. Он разработан в пику Java и JavaScript, чтобы придавать живое активное содержание статичным HTML-документам. Элементы языка в основном знакомы любому, кто программировал на процедурных языках с конструкциями типа `If...Then...Else`, `Do`, `While`, `For...Next loops` и т. д.

Что может VBScript

VB-сценарий обеспечивает довольно полную систему встроенных функций и команд, позволяя выполнять математические вычисления, программировать звуки, игры, создавать новые окна и новые URLs, контролировать действия пользователей при вводе данных в формы.

При этом код должен быть внедрен в страницу и выполнен, когда страница загружена. Вы можете также обрабатывать некоторые события, например активизировать код, когда нажимается кнопка в форме, или — печальный пример вируса ILOVEYOU! — после того, как вы выбрали и активизировали некоторый документ в списке. Сценарий (т. е. код на VBScript) может быть оформлен как файл с расширением .vbs и загружен к вам по сети из любого HTML-документа.

VBScript позволяет также устанавливать атрибуты, или реквизиты, средств управления OLE или апплетов Java, выполняющихся в окне просмотра. Этим способом вы можете легко изменять поведение дополнений (plug-ins) или других объектов, без необходимости глубокого постижения их внутренней структуры. Например, ваш код на VBScript мог бы автоматически начинать прокручивать внедренный AVI-файл, когда пользователь нажимает соответствующую кнопку.

Как VBScript встраивается в HTML-документ

Команды VBScript встраиваются в ваш HTML-документ либо непосредственно, либо через URL-ссылку, которая сообщает браузеру, откуда нужно загружать скрипт (в точности так же, как и в случае с JavaScript и другими

языками сценариев). Встроенный фрагмент VBScript заключается в контейнер HTML при помощи тегов `<SCRIPT>...</SCRIPT>`.

В элементе `<SCRIPT>` применяются два атрибута: `LANGUAGE`, который определяет выбранный язык сценариев, и `SRC`, специфицирующий URL, откуда загружается скрипт. Например:

```
<SCRIPT LANGUAGE="VBS|VBScript|Jscript|JavaScript">...</SCRIPT>
<SCRIPT SRC="http://www.rps.edu/scripts/common.VBS">...</SCRIPT>
<SCRIPT LANGUAGE="VBS" SRC="http://www.rps.edu//scripts/common.vbs">...
</SCRIPT>
```

VBScript похож на JavaScript и множество других компьютерных языков, но он имеет одно несомненное преимущество: на нем сразу без подготовки могут начинать программировать для Интернета все те, кто знаком с Visual Basic — самом популярном языке в мире!

Экспресс-описание VBScript

Использование идентификаторов

Идентификатор — это уникальное имя, которое VBScript использует для именования переменных, методов или объектов в программе. Все имена должны начинаться с алфавитного символа и могут содержать буквы и цифры. Их длина может составлять 255 символов, хотя на практике дело не доходит до 32 символов.

В отличие от JavaScript, который имеет литералы и переменные, VBScript применяет только переменные. Литералы можно использовать, просто присваивая значение переменным и не изменять их.

Переменные в VBScript могут получать значения любых поддерживаемых им типов данных. Среди этих типов следующие:

- `Integers` — может быть длиной 1, 2 или 4 байта;
- `Floating-Point` — VBScript поддерживает одинарную и двойную точность числа с плавающей запятой;
- `Strings` — строки могут быт представлены словами, фразами или произвольными данными и выделяются двойными кавычками;
- `Booleans` — логические переменные, принимают одно из двух возможных значений: "истина" или "ложь".

Использование встроенных объектов и функций

Индивидуальные элементы VBScript — это объекты. Например, литералы и переменные — это объекты типа переменные, которые могут использоваться для временного хранения данных различных типов. Любые объекты также имеют связанные с ними методы, оказывающие определенные действия на

различные типы данных. VBScript также дает возможность получать доступ к набору полезных объектов, которые представляет сам MS Internet Explorer, например, доступ к отображаемой странице и другим элементам сеанса просмотра.

Вы обращаетесь к объектам, определяя их имена. Например, активный объект документа именован как `document`. Чтобы использовать свойства документа или методы, вы должны добавить точку перед именем метода или свойства объекта, который вы хотите использовать. Например, `document.title` — это строка, в которой содержится заголовок документа.

Использование свойств

Каждый объект имеет устанавливаемые ему свойства. Например, чтобы узнать длину объекта типа строка, можно написать следующее выражение:

```
address.length
```

В ответ вы должны получить целое, в котором содержится количество символов в строке. Если используемый вами объект содержит свойство, которое можно модифицировать, вы запросто измените его:

```
house.color = "blue"
```

Вы можете также создавать новые свойства для объекта именуя их. Скажем, вы определяете класс, называемый `Customer`, для одной из ваших страниц. Новые свойства к объекту `Customer` добавляются следующим образом:

```
customer.name = "Иван Иванов"
customer.address = "Рабфаковская ул., 125"
customer.zip = "90210"
```

Чтобы добавить собственный метод к объекту `Customer`, например, метод с именем `Serg`, для начала надо написать функцию с именем `SergCustomer` и затем установить новое объектное свойство:

```
customer.Serg = SergCustomer;
```

Чтобы вызвать новый метод, вы должны написать следующее:

```
customer.Serg()
```

Выражения

Под *выражением* понимается то, что может быть вычислено и преобразовано к единственному значению. Выражение может содержать строки, числовые литералы, переменные, операторы и другие выражения и быть очень простым или чрезвычайно сложным.

Например, простое выражение:

```
sngPi = 3.14159
```

Или более сложное выражение:

```
(blnQuit = TRUE) And (blnComplete = FALSE)
```

Операторы

При помощи операторов совершаются действия над переменными, содержащими объекты различных типов. Операторы бывают двух типов.

- **Унарные (Unary).** Эти операторы требуют только одного операнда и оператор может ставиться либо перед, либо после операнда. Например, логический оператор `Not` является хорошим примером унарного оператора.
- **Бинарные (Binary).** Для данного типа оператора требуется два операнда. К этим типам операторам относятся следующие операции: (+) сложение, (-) вычитание, (*) умножение и (/) деление.

VBScript поддерживает оператор присваивания (=), операторы сравнения, такие как < и >=, логические операторы, такие как `And` и `Or`, и оператор конкатенации (&).

Операции управления в VBScript

Иногда сценарии, которые вы пишете, бывают очень просты и они, например, выполняют примитивные действия по загрузке и анимированию картинок. Однако можно написать и гораздо более искушенный сценарий с циклами управления и многочисленными подключаемыми процедурами. Если вы знакомы с любым другим машинным языком, вам, несомненно, будет знакомо большинство управляющих структур, используемых в VBScript. Они включают нижеприведенные типы.

- **Проверка условий.** Вы можете проверять выполнение условий в VBScript, используя структуру `If...Then...Elseif...Else...End`:

```
if (sngX > sngPi) then
    blnTest = TRUE
    intCount = intCount + 1
else
    blnTest = FALSE
    intCount = 0
end if
```

- **Повторяющиеся действия.** VBScript поддерживает несколько типов циклов, включая `For ... Next`, `For Each ... Next`, `While ... end`, `Do While ... End` и `Do ... Until`. Каждая из этих конструкций устанавливает собственные пределы. Например, ВЫПОЛНЯТЬ до тех пор, пока не наступит некоторое УСЛОВИЕ:

```
intCount = 1
do
```

```
    document.write "Count is " & CStr(intCount) & "<BR>"
    intCount = intCount + 1
  until (intCount = 101)
```

Примечание

Более подробную информацию обо всех аспектах программирования на VBScript можно получить по адресу **http://www.microsoft.com/vbscript**.

DHTML-скриптлеты

В основе скриптлета лежит идея сценария, что отражено в самом его названии, составленном из двух слов *scripting* и *applet*. Языки программирования сценариев, такие как Perl и Tcl, всегда были популярны у Unix-программистов, возможно, этим и воспользовалась Microsoft, активно превращая свой Visual Basic в язык сценарного типа — Visual Basic Script. Кроме того, в основе скриптлетов лежит объектная модель DHTML, а в будущем, по всей вероятности, будет использоваться DOM (спецификация W3 Document Object Model). В DHTML для программистов введено множество удобств для динамической обработки html-текстов. Например, посредством сценария можно обрабатывать любые имеющиеся теги, предварительно присвоив каждому из них свой идентификатор ID.

```
<P ID=MyText STYLE="color:blue">
```

Щелчком кнопкой мыши можно поменять цвет строки на зеленый, вставив следующий код на VBScript для обработки события:

```
Sub TheText_onClick()
  MyText.style.color="green"
End Sub
```

Согласно объектной модели документов (Document Object Model), браузер может сообщать о множестве различных событий, для обработки которых можно составить сценарии (табл. 4.1).

Таблица 4.1. *События и причины их возникновения*

Событие	Причина возникновения
OnMouseOver	Указатель мыши был подведен к элементу
OnMouseOut	Указатель мыши был отведен от элемента
OnMouseDown	Кнопка мыши была нажата, когда указатель находился в пределах элемента
OnMouseUp	Кнопка мыши была отпущена, когда указатель находился в пределах элемента

Таблица 4.1 *(окончание)*

Событие	Причина возникновения
OnMouseMove	Указатель мыши перемещался в пределах элемента
OnClick	Щелчок левой кнопкой мыши был выполнен, когда указатель находился на элементе
OnDblClick	Двойной щелчок левой кнопкой мыши был выполнен, когда указатель находился на элементе
OnKeyPress	Клавиша была нажата и отпущена. Если удерживать клавишу нажатой, запускаются несколько событий
OnKeyDown	Была нажата клавиша. Происходит лишь одно событие, вне зависимости от продолжительности удерживания клавиши
OnKeyUp	Клавиша была отпущена

Можно передавать события наверх ("bubble up") в контейнер, т. е. в более ответственную процедуру, связанную со всем документом.

Таким образом, в первом своем воплощении, скриптлетом можно было считать DHTML-страницу с прописанным в ней сценарием на VBScript или Jscript так, чтобы пользователь получил возможность настраивать эту страницу в соответствии со своими предпочтениями. В руки пользователя давались средства оформления и настройки таких страниц — это и был DHTML-скриптлет. В чем заключались основные преимущества оформления DHTML-страницы в манере скриптлета? Такая технология позволяла:

- с легкостью освоить DHTML, чем усердствовать в изучении трудного в понимании Java;
- повторно применять DHTML-страницы;
- использовать кроссплатформность, по крайней мере, в той сфере, где распространен применяемый браузер.

К недостаткам можно было отнести то обстоятельство, что даже при всем желании невозможно скрыть исходный текст скриптлета.

Для встраивания скриптлетов в исходную страницу Microsoft придумала новый MIME-тип, который направила в консорциум W3. В теге <OBJECT> он должен применяться следующим образом:

```
<object type="text/x-scriptlet" data="ScriptletUrl.html"></object>
```

Имя скриптлета в виде URL-ссылки указывается в параметре data. Главное, все-таки, было то, как следовало программировать скриптлет, чтобы пользователям были доступны его свойства, методы и события. То есть доступными становились атрибуты, которые всегда были присущи объектам, ибо скриптлет — это тоже объект.

Методы в скриптлете — это просто функции или процедуры, название которых начинается с префикса public_.

```
Sub public_Start()
  текст программы
End Sub
```

Если в скриптлете используется функция или процедура без приставки public_, значит она считается внутренней, т. е. private.

Свойства в скриптлете описываются с помощью префиксов public_get_ и public_put_. Например:

```
Sub public_put_AnimationSpeed(Speed)
```

Так устанавливается свойство скорости анимации.

Скриптлету можно передавать параметры:

```
<param name="ImageName" value="MyFirm.jpg">
<param name="Speed" value="100">
```

Что касается обработки событий, *то о них кратко уже говорилось в самом начале этого раздела.*

XML-скриптлеты

То, о чем говорилось выше, известно уже давно, а вот новая технология скриптлетов, о которой снова возвестила Microsoft, появилась сравнительно недавно. О чем речь? Microsoft разработала специальную DLL-библиотеку и предлагает теперь использовать для скриптлетов язык XML (eXtensible Markup Language). Это выглядит следующим образом: скриптлет является COM-компонентом, но реализует себя с помощью XML. На первый взгляд выглядит довольно загадочно, но если во всем разобраться, то становится постепенно понятно, что на XML описывается только структура скриптлета, т. е. определения его свойств, методов, обрабатываемых событий, а сам код, обеспечивающий функциональность, по-прежнему пишется на языке сценариев типа VBScript или JavaScript. Общие процедуры, используемые в Web-приложениях или на Web-сайтах, теперь могут быть преобразованы в скриптлеты. По сути, это настоящие повторно используемые компоненты, которые могут быть использованы на сервере (ASP-коды) или на клиенте, т. е. загружены в браузер.

Поскольку в данной технологии скриптлет объявляется полноценным COM-компонентом со всеми вытекающими отсюда возможностями, как-то:

- доступ к базам данных;
- возможность создания распределенной обработки;
- управление транзакциями.

Microsoft позаботилась о создании исполняющей машины для скриптлетов в виде сервера in-process (библиотека Scrobj.dll — входит в состав операционной системы Windows). Реализация всех сложных COM-интерфейсов, которые используют скриптлет, упрятана внутрь исполняющей машины, так что разработчику остается лишь разрабатывать функциональный код.

Файл, описывающий скриптлет, — это XML-файл со специальными тегами и расширением .sct. Скриптлет содержит следующие элементы:

- `<scriptlet>` — определение;
- `<registration>` — регистрация (если есть необходимость) скриптлета как COM-компонента;
- `<implements>` — объявления об используемых *interface handlers*, чтобы знать, к какому COM-компоненту будет принадлежать скриптлет;
- `<script>` — содержит код скриптлета.

```
<?scriptlet validate="true" error="true" debug="true"?>
   <scriptlet>
      <registration progid="progid" description="description"
                          version="version" clsid="{classid}">
      <implements type=COMhandlerName id=internalName>
         <property name="name"/>
         <method name="name"/>
      </implements>
      <script language="VBScript| JScript">
         Function method()
            'script here End Function
      </script>
      <script language="JavaScript">
         function method() {
            // script here
         }
      </script>
   </scriptlet>
```

Архитектура XML-скриптлетов

Если коротко определить суть XML-скриптлета, то можно сказать так: это COM-компонент, написанный на некотором языке сценариев. Вы можете написать его на Microsoft JScript®, Microsoft Visual Basic® Scripting Edition (VBScript) и любом другом языке, который содержит парсер, совместимый с ActiveX. Главная задача, решаемая при этом, формулируется таким образом, чтобы скрыть от пользователя всю сложность обычного написания COM-совместимых классов. В XML-скриптлетах активно используются библиотеки, входящие в состав COM+. При этом обеспечивается автоматическая

регистрация компонентов, диспетчеризация, управление соединениями и т. д. Другими словами XML-скриптлеты как бы имеют свою исполняющую машину и описывают объекты в манере, мало чем отличающейся от COM+. Должен быть обеспечен:

- уровень метаданных для описания интерфейсов;
- код для реализации методов.

Схематично архитектуру XML-скриптлетов можно изобразить, как показано на рис. 4.11. Скриптовый файл состоит из трех главных сегментов: *регистрирующая информация*, которая является общей для любого COM-компонента, *описание и отображение интерфейсов* и *код сценария*.

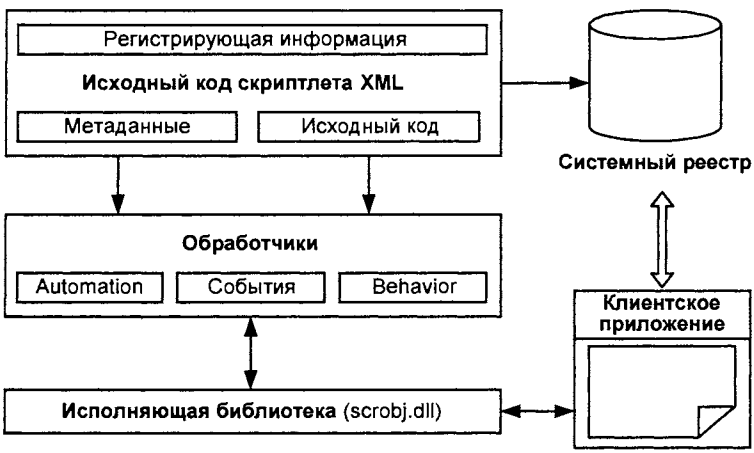


Рис. 4.11. Архитектура XML-скриптлетов

Клиентское приложение, если оно использует скриптлеты, всегда имеет дело с некоторой исполняющей системой (run-time scriplets) — системной DLL-библиотекой, называемой *scrobj.dll*. Она обеспечивает проводку, чтобы позволить любому COM-ориентированному клиентскому приложению использовать XML-скриптлет, как действительный двоичный COM-объект. Например, модуль run-time может экспозировать интерфейс `IDispatch` таким образом, чтобы клиенты могли его использовать. Тогда любая функция, вызываемая клиентом, будет принуждать модуль run-time вызывать соответствующий интерфейсный обработчик, к которому привязана порция кода JScript или VBScript. В этом случае мы действительно имеем дело с COM-объектом, написанным на простейшем языке сценариев. Исполняющая система обычно делает внешне доступными все или только те интерфейсы, для которых предопределены интерфейсные обработчики (interface handler) в исходном коде на XML. Каждый XML-скриптлет всегда имеет scrobj.dll в качестве встроенного модуля.

XML-скриптлет — это объект автоматизации (Automation object), в котором нет пользовательского интерфейса (UI). Если вам необходим такой интерфейс, желательно использовать DHTML-скриптлеты.

Типичным случаем применения XML-скриптлетов является разработка небольших, гибких, повторно-используемых компонентов для запуска их на Web-серверах из ASP-страниц *(см. далее раздел "ASP-сценарии")*.

Примеры XML-скриптлетов

Регистрация и описание

Как уже говорилось выше, исходный код XML-скриптлета состоит из 3-х частей, оформленных в виде соответствующих тегов: `<registration>`, `<implements>` и `<script>`. Первый из них, `<registration>`, обеспечивает некоторое описание скриптлета — `progid`, версия `CLSID`. `CLSID` может быть сгенерирован при помощи специальной утилиты или мастера XML Scriptlet Wizard (Microsoft). Пример такого описания:

```
<registration
   description="Мой первый XML-скриптлет"
   progid="First.Scriptlet"
   version="1.00"
   classid="{11CA6520-5EE0-11D2-BC00-000005000E67}">
</registration>
```

Файл, в котором содержится XML-скриптлет имеет расширение .sct. После того как вы его создали посредством любого текстового редактора, не забудьте зарегистрировать с помощью программы regsvr32.exe или, переместив на имя этого файла указатель мыши, нажмите правую кнопку мыши и выберите пункт **Register**.

Программный интерфейс

Тег `<implements>` определяет COM-интерфейс XML-скриптлета. На сегодня имеется немного интерфейсов, которые могут быть выполнены: только те, для которых существует обработчик интерфейса. Обычно требуются интерфейсы для объектов типа `Automation`, обработки различных событий и использования IE5. Вот пример:

```
<implements type="Automation">
  <method name="GetDataFromSource">
    <parameter name="providerName" />
    <parameter name="cmdStr" />
  </method>
  <method name="ConvertDataToXML" />
</implements>
```

Этот скриптлет демонстрирует два своих метода GetDataFromSource и ConvertDataToXML . Следующие два параметра определяют имя провайдера и командную строку для выполнения запроса. Второй метод вообще не требует никаких аргументов.

Пример конкретного скриптлета, выполняющегося на сервере и извлекающего некоторые данные из некоторой таблицы в СУБД, приводится ниже. Но прежде, чем вникать в его текст, рекомендуем ознакомиться с разделом настоящей главы, специально посвященной ASP-сценариям.

Пример XML-скриптлета для работы с таблицей данных

```
<?XML version="1.0"?>
<scriptlet>
<registration
   description="Доступ к любой таблице в СУБД"
   progid="Server.GetContacts"
   version="1.00"
   classid="{8D9CC880-D79F-11d2-B7C8-00C0DFE39736}"
>
</registration>
<public>
   <method name="GetRecords">
      <parameter name="conn" />
      <parameter name="cmdText" />
   </method>
</public>

<script language="VBScript">
<![CDATA[
'// GetRecords метод
Function GetRecords (conn, cmdText)
  Set rs = CreateObject("ADODB.Recordset")
  rs.CursorType = 3     'adOpenStatic
  rs.Open cmdText, conn

  Set GetRecords = rs
End Function
]]>
</script>
</scriptlet>
```

Данный скриптлет получает на входе 2 параметра: строку связи с базой данных и строку запроса, определяемых в методе GetRecords. Для доступа к данным используется ADO-интерфейс, который определяется в функции,

реализующей метод. Тега `<implements>` здесь вообще нет, хотя можно было бы определить его, как `<implements type="ASP">`.

Теперь, чтобы вызвать этот скриптлет из HTML-страницы, подготовьте страницу, описанную в следующем примере.

Пример ASP-страницы, в которой вызывается XML-скриптлет

```
<HTML>
<HEAD>
<TITLE>Empl.asp Example</TITLE>
</HEAD>
<BODY bgcolor="lightblue">
<H1> Служащий #1 </H1>
<%
' Создание объекта по имени зарегистрированного скриптлета
Set oData = CreateObject("Server.GetContacts")
' Передача в скриптлет 2-х параметров: имени ODBC-источника и строки
  запроса
Set rst = oData.GetRecords("WebMeals", "SELECT * FROM qryCurrentEmployees
WHERE EmployeeID = 1")
%>
' Показ данных на экране дисплея
<B>Имя:</B> <%= rst("FirstName") %>
   <%= rst("LastName") %>
<BR><B>Адрес:</B> <%= rst("Address") %>
<BR><B>Город/Область/Код:</B> <%= rst("City") %>,
 <%= rst("State") %>
   <%= rst("ZipCode") %>
<%
' Закроем все объекты
rst.Close
%>
</BODY>
</HTML>
```

На экране вы увидите то же изображение, что и в случае выполнения обычной ASP-страницы с доступом к базе данных, когда будете читать об этом чуть ниже. Тут дело вкуса — использовать скриптлеты или нет? На мой взгляд, они улучшают функциональность программы и придают ей компонентность.

Java-скриптлеты

Подробное описание Java-скриптлетов изложено ниже в разделе *"JSP-сценарии"*, где в деталях излагается концепция JSP-технологии.

ASP-сценарии

ASP (Active Server Pages) — это, пожалуй, самая популярная технология Microsoft, предназначенная для разработки эффективных Интернет-приложений и оживления Web-страниц различными динамическими данными, получаемых из самых разнообразных источников. Грубо говоря, ASP — это все те же VBScript-коды, только они не загружаются в браузер и не интерпретируются им, а выполняются непосредственно на сервере в контексте MS IIS (Microsoft Web Internet Infotmation Server). Раньше необходимо было провести дополнительную инсталляцию специального 10-мегабайтного файла, чтобы модифицировать существующий IIS, входящий в состав операционной системы Windows NT 4.0, но начиная с Win2K, поддержка ASP в IIS стала стандартной. Текстовые файлы с расширением .asp — это обычные HTML-файлы с вставленными в них кодами на VBScript. В отличие от обычных кодов, в них можно применять многочисленные встроенные методы, которые резко облегчают процесс программирования и используют объектную модель и объектно-ориентированные методы. ASP-поддержка в IIS выстроена в точном соответствии с 3-уровневой моделью клиент-серверных технологий (рис. 4.12).

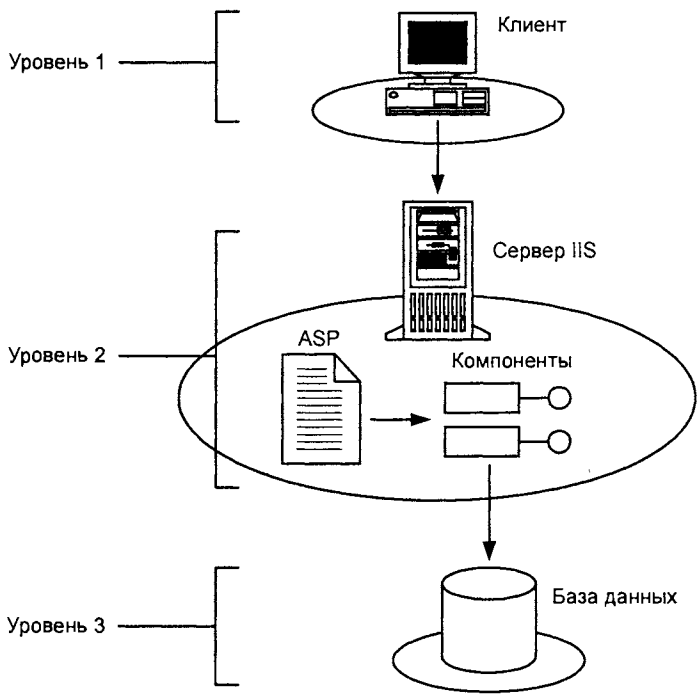


Рис. 4.12. Использование 3-уровневой модели для ASP

Распределенная архитектура Windows DNA

Как уже говорилось в предыдущих главах, Microsoft разработала для Интернета особую распределенную архитектуру — Windows DNA (Distributed Internet Application Architecture) — как способ интеграции Web в многоуровневую модель разработки.

Windows DNA является Интернет-интеграцией всех предыдущих технологий Microsoft, таких как:

- Dynamic HTML (DHTML);
- Active Server Pages (ASP);
- COM-компоненты;
- служба каталогов (Active Directory);
- механизмы защиты (Windows® Security Services);
- управление очередями сообщений (Microsoft® Message Queuing);
- доступ к разнообразным источникам данных (Microsoft Data Access Components).

Microsoft разработала архитектуру Windows DNA, опираясь только на открытые стандарты и протоколы, гарантируя тем самым ее широкое и повсеместное использование тысячами других разработчиков.

Архитектура Web-сервера Microsoft IIS

IIS является важной составной частью архитектуры Windows DNA. Важная роль IIS обусловлена доступом клиентов по протоколу HyperText Transfer Protocol (HTTP) к другим ключевым сервисам и службам Windows DNA, таким как DHTML, ASP и т. д. (рис. 4.13).

Чтобы читателю было понятнее, что изображено на рисунке, стоит немного пояснить.

Существует достаточное количество различных методов создания программ CGI/ISAPI/NSAPI для серверов приложений. Эти программы перебрасываются на Web-сервер и выполняются под его непосредственным управлением. Главное их отличие заключается в том, что CGI-программы запускаются в виде отдельных процессов, а программы ISAPI и Netscape(NSAPI) выполняются в контексте Web-сервера. В основе этих методов лежит идея преобразования CGI-процесса в некое подобие DLL. Дело в том, что DLL существуют в двух видах:

- `Load-time dynamic linking` — при явном вызове DLL-функции этот метод требует, чтобы приложение было построено таким образом, чтобы связать импортируемую функцию с выполнимым кодом еще до старта программы;

- `Run-time dynamic linking` — когда используются функции `LoadLibrary` и `GetProcAddress`, чтобы получить адрес импортируемой DLL-функции после старта программы. Этот метод не требует дополнительной компоновки.

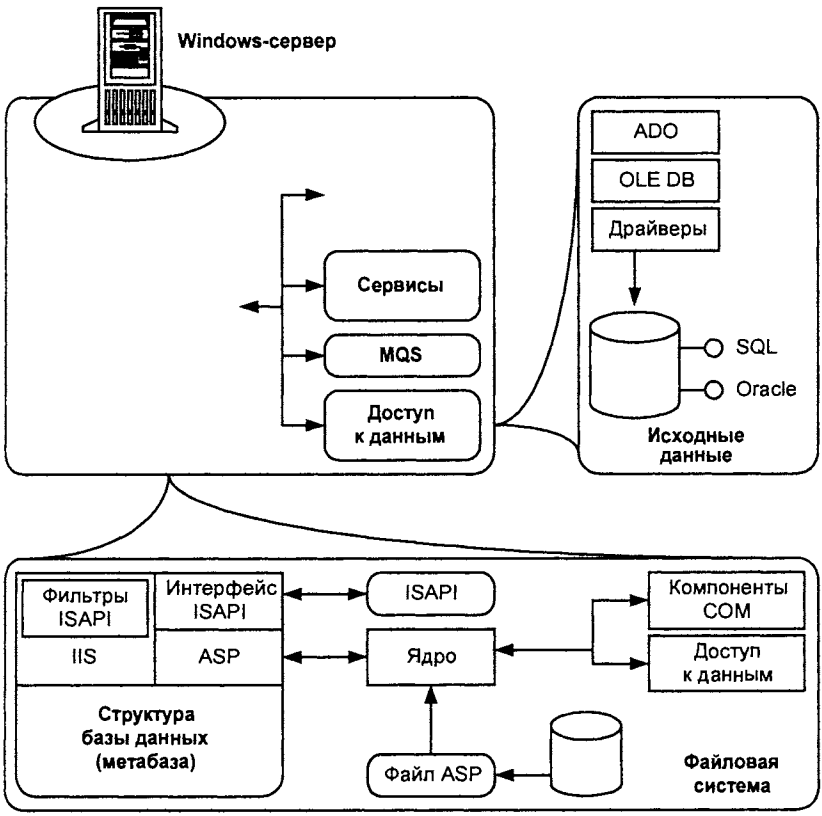


Рис. 4.13. Общая архитектура Web-сервера Microsoft

В отличие от EXE-программы, ISAPI-программы загружаются в одно и то же пространство Web-сервера. Когда сервер загружает процесс, он передает управление в точку `GetExtensionVersion`. При каждом запросе клиента сервер передает управление в точку `HttpExtensionProc`.

Для каждого клиента при этом создается ECB (Extension Control Block) — управляющая структура. В этой управляющей структуре имеется уникальный идентификатор, характеризующий клиента и по которому будет производиться обмен данными с ним.

Приведем наиболее важные поля этой структуры:

- `cbSize` — размер структуры в байтах;
- `ConnID` — идентификатор канала (не изменяется!);

- `dwHttpStatusCode` — для записи результата завершения операции (код состояния транзакции. Код успешного завершения — 200 (см. спецификацию протокола HTTP);
- `lpszMethod` — имя метода (эквивалентно переменной среды с именем `REQUEST_METHOD`);
- `cbAvailable` — размер блока данных, полученных от пользователя;
- `lpbData` — указатель на область памяти, в которую записан сервером полученный от пользователя блок данных;
- `lpszContentType` — тип принятых данных, например, `text/html`;
- `GetServerVariable` — указатель на функцию, с помощью которой можно прочитать переменные среды окружения;
- `WriteClient` — посылка данных клиенту;
- `ReadClient` — с помощью этой функции можно дочитать дополнительные данные, не поместившиеся в буфер предварительного чтения, имеющий адрес `lpbData`.
- `ServerSupportFunction` — посылка стандартных заголовков HTTP и др.

Обмен данными идет при помощи функций `ReadClient`, `WriteClient`. Получить информацию о соединении можно по функции `GetServerVariable`.

ISAPI и NSAPI (Netscape) — это DLL-библиотека, которая загружается Web-сервером по запросу клиента.

CGI-приложение — это консольное приложение, которое получает информацию по запросу клиента из стандартного ввода и выводит результаты на стандартный вывод.

WinCGI — это Web-приложение в виде Windows-приложения, которое получает информацию по запросу клиента из конфигурационного INI-файла, записанного на сервере, а результаты также передаются в файл, который отправляется клиенту.

Отличительной особенностью и преимуществом разработки Web-модулей является тот факт, что они могут располагаться и выполняться на Web-серверах, где установлены ODBC-драйверы для доступа к базам данных. Тем самым достигается уровень ультратонкого клиента на рабочих местах, где достаточно установить только браузер, который по сути играет роль программного интерфейса к модулю, выполняющему полезную работу на удаленной машине. Это как нельзя лучше отвечает идее использования узлов концентрации под управлением Windows NT.

Клиент запускает программу из сетевого навигатора, связываясь с промежуточным Web-узлом, где выполняется Web-модуль. Этот модуль, в свою очередь, обращается к серверу базы данных Unix или Windows NT через ODBC и работает с соответствующими реляционными таблицами, обеспечивая выбор пользователя, непосредственно передавая данные соответствующих запросов в браузер, автоматически перекодируя их в формат HTML.

Преимущества ASP-программирования совершенно очевидны, потому что пользователь-клиент может работать с браузером как с окном, сквозь которое он может запускать на выполнение мощные полнофункциональные Web-приложения и получать результаты их выполнения на сервере в виде обычных HTML-файлов. В данном случае четко разделяется логика клиента и логика сервера и профессиональный программист, создающий многопользовательское ASP-приложение, может воспользоваться всей мощью машины-сервера, для которого данное приложение создается. Клиент может почувствовать эту мощь, запустив у себя окно, в котором будет функционировать ASP-приложение, т. е. он может использовать мощь сервера виртуально.

В качестве примера дадим несколько ASP-программ, демонстрирующих их возможности при работе с базами данных.

Прежде чем начать программировать, лучше всего прочитать несколько страниц великолепной документации Microsoft, входящей в состав Windows 2000 и посвященной программированию для IIS (**http://localhost/iisHelp/iis/misc/default.asp**).

При программировании с использованием ASP применяется термин *компоненты*.

По терминологии Microsoft *компонентом* называется исполняемый код, существующий в виде библиотеки DLL или EXE-программы. Компонент может содержать один или более объектов, выполняющих некоторые функции. Чтобы использовать эти готовые объекты в ASP-сценариях, нужно создать экземпляр данного объекта и присвоить его значение некоторой переменной. Для этого применяется метод `Server.CreateObject` или тег HTML `<OBJECT>`. Например, если программа имеет зарегистрированный программный идентификатор `PROGID "MSWC.AdRotator"`, то можно создать экземпляр объекта `AdRotator`, используя следующий оператор:

- **VBScript:**

```
<% Set MyAds = Server.CreateObject("MSWC.AdRotator") %>
```

- **JScript:**

```
<% var MyAds = Server.CreateObject("MSWC.AdRotator") %>
```

Поскольку мы будем использовать метод доступа к данным ADO, то создадим свой объект под именем `ADODB.RecordSet`.

Объектный интерфейс Microsoft на базе OLE DB — ADO

Основным недостатком всех прошлых решений в части разработки приложений для Интернета был их громоздкий интерфейс для доступа к базам данных, основанный на JDBC/ODBC, и неудобные средства визуализации

данных, получаемых из баз. Предложив в качестве универсального интерфейса для доступа к различным источникам данных OLE DB, а в качестве интерфейса к интерфейсу — удобную и компактную модель ADO (Active Data Objects), Microsoft разом решила трудные задачи и поставила многих программистов перед нелегким выбором — осваивать им Visual Basic или нет? Дело в том, что если соблазниться удобными и привлекательными сторонами Web-сервера Internet Information Server 3.0/4.0, то поневоле нужно выбирать между JavaScript и Visual Basic Script, а уж сделав выбор в пользу последнего, придется полностью забыть о Netscape Navigator и предпочесть ему IE 4.0/5.0. Многие программисты пошли именно этим путем, вовсе не оттого, что они внезапно разлюбили Netscape, нет,— они вынуждены были сделать это из-за того, что многие вещи в Netscape стали трудновыполнимыми, особенно когда приходится посещать сайты с ASP-страницами. Кроме того, ADO-интерфейс лучше и проще всего реализован в VB и в его диалекте VBScript — основном языке сценариев MS Internet Explorer.

Прежде чем перейти к описанию ADO, вначале, все-таки, вспомним наиболее характерные черты интерфейса OLE DB, а затем уже перейдем к конкретным примерам программирования с использованием ADO.

ODBC, OLE DB и ADO

ODBC и OLE DB — это прикладные интерфейсы Windows для осуществления доступа к данным. Более старая спецификация ODBC обеспечивает доступ к данным и, прежде всего, к реляционным базам, основанным на использовании языка SQL. OLE DB — это спецификация следующего поколения Microsoft, которая позволяет осуществлять доступ к данным через так называемых *провайдеров*, т. е. поставщиков данных, тем самым делая круг данных намного шире. Эти поставщики могут включать нереляционные системы баз данных, системы электронной почты и системы CAD/CAM, а также классические реляционные системы баз данных.

OLE DB не заменяет ODBC. Фактически, OLE DB включает поставщика данных, который позволяет вам применять новый стандарт вместе с традиционными ODBC-источниками данных. Важно отметить, что Microsoft планирует использовать OLE DB в качестве универсального стандарта для доступа к любым данным, независимо от того, где или как они сохраняются на предприятиях.

Как ADO вписывается в эту картину? ADO — это интерфейс высокого уровня к стандарту OLE DB. Вы можете умело применять его в своих программах, чтобы получать доступ через ODBC-источники к провайдерам данных OLE DB. Пока что, чаще всего используются источники, указывающие на реляционные данные. В будущем, однако, вы сможете применять ADO, чтобы обратиться к вашей системе электронной почты или некоторому другому нереляционному поставщику данных.

Microsoft публично декларировала, что ADO в конечном счете заменит мириады разнообразных моделей доступа к данным различных компаний, включая методы самой Microsoft, такие как DAO и RDO, так что имеется в виду, что ADO — это не только метод для реализации Интернет/интранет-доступа к данным. ADO — это модель доступа к данным, предлагаемая Microsoft в будущем!

Объектная модель ADO

Модель объекта ADO, показанная на рис. 4.14, является намного более простой, чем модели объекта DAO и RDO.

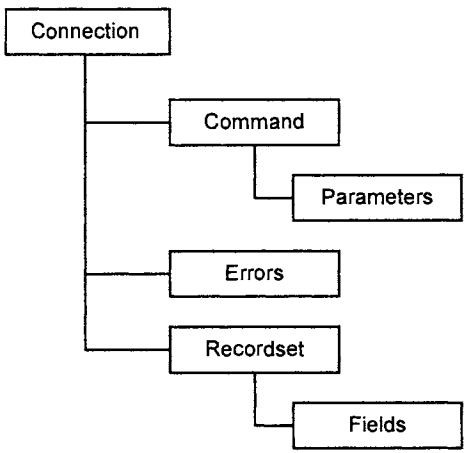


Рис. 4.14. Верхний уровень в ADO представляет объект `Connection`, который содержит объекты `Command` и `Recordset`, а также совокупность ошибок

Как вы можете видеть на рисунке, ADO, в отличие от DAO и RDO, не содержит ни совокупности таблиц, ни среду, ни рабочую область, ни исполняющую машину баз данных. Главные объекты ADO следующие:

- **`Connection`** — вы можете использовать объект ADO `Connection`, чтобы создать соединение с провайдером данных. Объект ADO `Connection` подобен DAO-объекту базы данных или объекту RDO `rdoConnection`.
- **`Recordset`** — вы можете использовать объект ADO `Recordset`, чтобы создать набор записей из запроса. Подобно DAO `Recordset` и RDO `rdoResultset`, вы можете двигаться вперед и назад через ADO `recordsets`. Иногда, ADO `recordsets` называются *курсорами*. Курсоры ADO `recordset` всегда строятся на сервере.
- **`Command`** — вы можете использовать объект ADO `Command`, чтобы указать на SQL-строки, сохраненные процедуры или запросы (queries), которые вы можете выполнять. Объект `Command` подобен объекту DAO `QueryDef` или RDO `rdoPreparedStatement`.

- `Errors collection` — совокупность ошибок ADO позволяет вам циклически обрабатывать ошибки, которые могут происходить из-за неудачной попытки доступа к данным. Поскольку один оператор обращения к данным может производить многократные ошибки, ADO определяет вместо одиночной ошибки целую совокупность ошибок, ориентируясь на возможности, заложенные в стандарте OLE DB, на который полностью опирается этот высокоуровневый интерфейс.

При желании ADO можно рассматривать как модель доступа к данным (интерактивный язык программирования для взаимодействия с базами данных), оптимизированных для использования в Интернете/интранете. Если вы знакомы с другими моделями данных Microsoft, т. е. уже использовали DAO в MS Access или RDO в Visual Basic, то вы будете чувствовать себя вполне уверенным, осваивая ActiveX Data Objects (ADO). Хотя вы можете найти некоторые различия между моделями доступа к данным, вероятнее всего, что вы, все-таки, найдете больше подобий, чем различий.

Теперь, все-таки, вернемся к ASP-сценариям и посмотрим, каким образом можно использовать ADO в ASP? Прежде чем начать программирование, необходимо, чтобы ADO был установлен на вашей машине. Вы можете приобрести ADO, установив Active Server Pages, Visual InterDev или OLE DB SDK на вашей машине. Если вы уже работаете в операционной системе Windows 2000, то ничего инсталлировать не надо!

Примеры, приведенные далее, требуют, чтобы вы имели установленный MS SQL Server.

Одни из примеров требуют доступа к SQL-серверу, другие используют данные, сохраненные в базе данных MS Access.

ASP-примеры

Примеры подготовлены по материалам конференции *MS Office and VBA Solution* (Лондон, 12-15 июля 1998 — сообщение и публикация Пола Литвина (Paul Litwin)).

- Создайте новый каталог на вашем Web-сервере и разрешите запросы к нему на чтение и выполнение. Скопируйте ASP- и HTML-файлы в этот каталог.
- Создайте ODBC-источник данных, именованный как WebSQL, который будет указывать на версию базы данных Pubs на сервере MS SQL с таблицей `Employees`.
- Создайте ODBC-источник данных, именованный WebBase, который будет указывать на версию базы данных Emps в MS Access 97, содержащей таблицу служащих с именем `qryCurrentEmployees`.
- Вызовите из браузера соответствующую HTML-страницу, например, **http://server_name/directory_name/emp1.asp**, где **server_name** — имя вашего Web-сервера и **directory_name** — имя каталога Web, который вы создали.

ADO-доступ к данным

Следующий ASP-файл (emp1.asp) использует ADO, чтобы получить запись из базы данных Emps.mdb (MS Access-97), описанной через источник WebBase ODBC.

Распечатка файла emp1.asp:

```
<HTML>
<HEAD>
<TITLE>Emp1.asp Example</TITLE>
</HEAD>
<BODY bgcolor="lightblue">
<H1> Служащий #1 </H1>
<%
' Объявить переменные
Dim rst
Dim varSQL
' Создать пустой объект recordset
Set rst = Server.CreateObject("ADODB.Recordset")
' Создать SQL-оператор и перенаправить вывод его в переменную varSQL
varSQL = "SELECT * FROM qryCurrentEmployees " & _
 "WHERE EmployeeID = 1"
' Заполнить recordset записями, полученными
' по SQL-запросу к источнику WebBase
rst.Open varSQL, "WebBase"
' Показать данные
%>
<B>Имя:</B> <%= rst("FirstName") %>
   <%= rst("LastName") %>
<BR><B>Адрес:</B> <%= rst("Address") %>
<BR><B>Город/Область/Код:</B> <%= rst("City") %>,
 <%= rst("State") %>
   <%= rst("ZipCode") %>
<%
' Очистить все объекты
rst.Close
%>
</BODY>
</HTML>
```

То, что получается при выполнении этого кода, изображено на рис. 4.15.

Внимание!

Если вы хотите выполнить этот пример, то должны создать источник данных, называемый WebBase, на вашем Web-сервере, который указывает на базу данных EMPS.mdb.

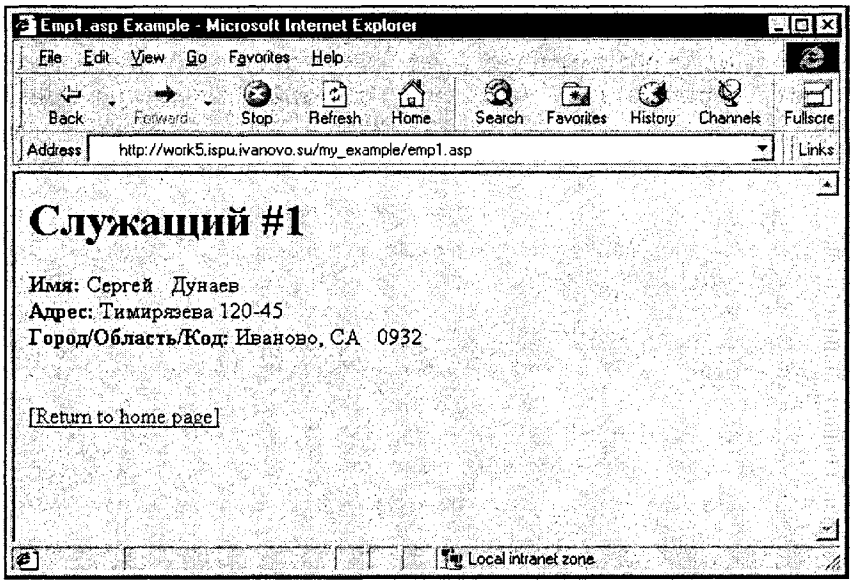


Рис. 4.15. Эта страница использует ADO-код, чтобы извлечь одиночную запись из запроса к таблице `qryCurrentEmployees` и отобразить ее на Web-странице

Чистый HTML. Поначалу кажется, что файл emp1.asp содержит множество строк кода, но фактически все совершенно просто, если вы мысленно разбиваете его на несколько меньших частей. Первая часть ASP-страницы — только чистый HTML-код:

```
<HTML>
<HEAD>
<TITLE>Emp1.asp Example</TITLE>
</HEAD>
<BODY>
<H1> Employee #1 </H1>
```

Объявление переменных. В этом куске кода объявляются две переменные. Не забывайте, что все переменные являются вариантами VBScript (язык, принятый по умолчанию в ASP), так что здесь используется описание `DIM` и не используется `AS`.

```
<%' Объявить переменные
Dim rst
Dim varSQL
```

Создание объекта `Recordset`. Вы можете создать набор записей `recordset` при помощи нескольких шагов:

1. Создайте пустой объект `Recordset`:

```
Set rst = Server.CreateObject("ADODB.Recordset")
```

Если вы уже используете DAO или RDO в VBA, этот путь создания объектов может показаться странным. VBA позволяет раннее связывание объектов. Вы можете установить ссылку на объект в библиотеке типов объектной модели. (Иногда эта ссылка создается автоматически.) Поэтому, чаще всего, программисты избегают необходимости использовать функцию `CreateObject`. К сожалению, поскольку язык VBScript не поддерживает раннее связывание, вы должны создать все объекты с помощью функции `CreateObject`.

Вы создаете объекты ASP, которые используют объект `Server`, для обеспечения доступа к методам и свойствам Web-сервера.

2. Добавьте операторы SQL в переменную `varSQL`:

```
varSQL = "SELECT * FROM qryCurrentEmployees " & _
"WHERE EmployeeID = 1"
```

Для источников данных MS Access, вы можете обращаться как к таблицам, так и к запросам при помощи ваших операторов SQL.

3. Используйте метод `Open` в объекте `Recordset`, чтобы заполнить его записями, возвращаемыми при выполнении оператора SQL из переменной `varSQL`.

Второй параметр, в этом примере `WebBase`, сообщает ADO имя источника данных, которое необходимо использовать:

```
rst.Open varSQL, "WebBase"
%>
```

Показ данных. Чтобы отобразить данные из записи, возвращенной объектом `recordset`, вы должны использовать смесь кодов HTML и VBScript. HTML обеспечивает метки для данных, а VBScript — значения из `recordset`, используя имена полей из запроса MS Access:

```
<B>Name:</B> <%= rst("FirstName") %>
   <%= rst("LastName") %>
<BR><B>Address:</B> <%= rst("Address") %>
<BR><B>City/State/Zip:</B> <%= rst("City") %>,
 <%= rst("State") %>
   <%= rst("ZipCode") %>
```

Вы можете вставлять непрерывающийся пробел в ваш HTML-код, используя `$nbsp`; последовательность символов.

Очистка. Последний раздел кода закрывает объект `Recordset` и заканчивает HTML-код:

```
<%
rst.Close
%>
```

```
</BODY>
</HTML>
```

Все это не настолько плохо, не правда ли?

Простая объектная модель

В ADO объектная иерархия имеет меньшее количество ветвей. Это означает, что, в отличие от DAO и RDO, вы не должны спускаться по стволу дерева иерархий вниз только для того, чтобы добраться до объекта. Это преимущество демонстрируется в Emp1.asp, где непосредственно можно создать `rst.recordset` без необходимости создания соединения.

В самом деле, это является заметным преимуществом модели, потому что расходуется несколько меньше памяти и используется меньшее количество строк кода. Всегда сохраняется больше памяти, если избегать создания объекта `Connection`. Использование памяти является критическим ресурсом, если сотни клиентов одновременно используют вашу Web-страницу.

Во многих случаях, однако, вы должны явно создать объект `Connection`, потому что это более эффективно, когда вы выполняете множество наборов `recordset` или командных операций.

Опции объекта *Recordset*

Метод `Open` в `Recordset` позволяет вам создавать различные виды наборов. Рассмотрим синтаксис подробнее:

```
recordset.Open source, activeconnection, _
cursortype, locktype, options
```

Первый аргумент метода `Open` представляет источник `recordset`. В этом случае, можно использовать SQL-строку как в emp1.asp, но можно также использовать имя таблицы или сохраненной процедуры на SQL-сервере.

`ActiveConnection`. Во втором параметре, `ActiveConnection`, вы можете сделать один из следующих вариантов:

- использовать строку соединения, которая создает новое соединение для единственной цели — создания `recordset`;
- указать на активный объект `Connection`, который вы уже создали.

`CursorType`. Вы используете третий параметр, `CursorType`, чтобы определить тип набора записей `recordset`, который создаете. Возможные значения этого параметра показаны в табл. 4.2.

В отличие от наборов данных DAO, вы можете модифицировать все наборы данных ADO, даже статический и "только вперед". Различные типы наборов данных варьируются в зависимости от того, как отражаются изменения, сделанные другими пользователями.

Таблица 4.2. *Значения параметра* `CursorType`

Значение	Константа	Описание
0	adOpenForwardOnly	Набор записей recordset, который позволяет перемещаться только вперед (Forward-only recordset cursor). Используется по умолчанию. Не поддерживает методы типа MovePrevious, MoveFirst и MoveLast, а также использование свойства RecordCount
1	adOpenKeyset	Ключевой набор recordset (Keyset recordset cursor) очень напоминает DAO dynaset. Число записей в этом наборе никогда не изменяется. Вы не сможете узнать относительно добавленных другими пользователями записей, и удаленные ими записи не исчезают из поля вашего зрения (хотя удаленные записи становятся недоступными). Однако вы получаете сообщения, когда существующие записи, с которыми вы работаете, кем-то модифицируются
2	adOpenDynamic	Динамический набор записей recordset (Dynamic recordset cursor) не имеет никакого DAO-дубликата. Этот тип полностью динамический. Когда вы открываете такой набор данных, то получаете сообщения относительно добавлений, удалений и модификаций записей. Динамический recordset наиболее функционален, но также и наиболее требователен в отношении памяти и быстродействия
3	adOpenStatic	Статический набор записей recordset (Static recordset cursor)

Изменения в записях, сделанных другими пользователями в статическом и "только вперед" наборах данных, невидимы для вас. То есть, когда вы создаете статический или "только вперед" recordset, вы не задумываетесь об изменениях, сделанных другими пользователями. Если вы не должны двигаться назад по набору данных, тогда используйте recordset "только вперед", потому что он потребляет меньшее количество памяти и быстрее, чем статический набор.

LockType. Четвертый параметр определяет тип блокировки. Возможные типы блокировок показаны в табл. 4.3.

Установите параметр LockType в значение 1, если вы не модифицируете записи, тогда записи станут доступным только для чтения. Если вы модифицируете записи, то следует установить LockType в значение 3 (оптимисти-

ческая блокировка). Все провайдеры данных поддерживают этот тип блокировки. Когда вы выбираете оптимистическую блокировку, запись блокируется только в то время, когда ADO физически модифицирует запись. Напротив, параметр `Locktype` со значением `2` обеспечивает пессимистическую блокировку (если провайдер данных поддерживает ее). В этом типе блокировки запись блокируется, когда вы ее редактируете. Некоторые провайдеры данных не обеспечивают пессимистическую блокировку.

Таблица 4.3. *Типы блокировок*

Значение	Константа	Описание
1	`adLockReadOnly`	Только чтение (Read only)
2	`adLockPessimistic`	Пессимистическая блокировка (Pessimistic locking)
3	`adLockOptimistic`	Оптимистическая блокировка (Optimistic locking)
4	`adLockBatchOptimistic`	Оптимистическая блокировка пакета данных (Optimistic locking with batch updates)

Установка параметра `LockType` в значение `4` говорит ADO о том, что вы хотите выполнить корректировку пакета записей. Эта форма модифицикации позволяет вам выгружать множество записей, модифицировать их локально и посылать обратно провайдеру в едином пакете. Большинство провайдеров данных не поддерживает пакетное модифицирование.

Опции. Дополнительно вы можете использовать параметр `Options`, чтобы указать тип параметра `Source`, который вы обеспечили. Тип опций показан в табл. 4.4.

Таблица 4.4. *Типы опций*

Значение	Константа	Описание
1	`adCmdText`	SQL-оператор (Command text (SQL statement))
2	`adCmdTable`	Таблица, вид или запрос (Table, view, or saved select query)
4	`adCmdStoredProc`	Сохраненная процедура или сохраненный запрос (Stored procedure or saved action query)
8	`adCmdUnknown`	Неизвестный тип (Unknown). Используется по умолчанию

Если вы посылаете SQL-оператор, как параметр `Source`, тогда установите `Options` в значение `1`. Если параметр `Source` указывает на имя таблицы, то

нужно выбрать запрос или представление (вид) и затем использовать значение 2. Не все провайдеры данных позволяют использовать опцию adCmdTable.

Если вы хотите вызвать сохраненную процедуру или сохраненный запрос, тогда установите Options в значение 4.

Что такое константы? Константы, перечисленные в вышеприведенных таблицах, не предопределены для VBScript, как это сделано, когда вы используете ADO из VBA (по отношению к библиотеке типов ADO). Вы можете включить эти константы в ваш код двумя способами:

- определить свои собственные константы, использующие предложенные имена;
- включить файл adovbs.inc в исходный код. (Если вы используете JavaScript, рекомендуется включать файл adojavas.inc вместо adovbs.inc.)

Эти файлы можно найти в каталоге Files\Common Files\System\Ado на Web-сервере. (Более старые версии Microsoft Web помещают эти файлы в каталог \Inetpub\wwwroot\ASPSamp\Samples.)

Чтобы включить Include-файл в ваш ASP-файл, используйте на сервере, где будет выполняться ASP-файл, такой оператор:

```
<! – *include virtual= "/ASPSAMP/SAMPLES/ADOVBS.INC" – >
```

Вы должны поместить этот оператор вне любого сценария. Например, рассмотрим следующий блок кода:

```
<%
Dim con
Dim rst
Dim strSQL
%>
<!--#include virtual="/ASPSAMP/SAMPLES/ADOVBS.INC"-->
<%
Set con = Server.CreateObject("ADODB.Connection")
' ...
```

Разумеется, можно и вручную определить константы, если вы нуждаетесь всего в нескольких из них, тем более, что файл advobs.inc может потребовать значительный блок памяти.

Вывод множества записей. Пример emp1.asp выводит одну запись на Web-страницу. Однако чаще всего надо отображать группу записей в HTML-таблице, как показано во втором примере, cust1.asp. Второй пример выводит записи из SQL-версии базы данных PUBS (таблица Employees).

Следующий пример является полным ASP-файлом:

```
<HTML>
<HEAD>
```

```
<TITLE>Cust1.asp Example</TITLE>
</HEAD>
<BODY>
<H1> Наши пользователи </H1>
<%
' Объявить переменные
Dim con
Dim rst
Dim strSQL
' Объявить константы для  метода Open
Const adOpenStatic = 3
Const adLockReadOnly = 1
Const adCmdText = 1

' Создать Connection-объект и указатель
' к WebSQL-источнику данных с
' пользовательским именем и паролем
Set con = Server.CreateObject("ADODB.Connection")
con.Open "WebSQL", "Employees", ""
' Создать Recordset-объект и заполнить его
' записями из SQL-оператора, используя
' уже установленное соединение.
' Пока эта страница используется для создания HTML-таблицы,
' применяется static, read-only recordset
strSQL = "SELECT * FROM Employees " & _
 "ORDER BY LastName, FirstName"
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open strSQL, con, adOpenStatic, _
 adLockReadOnly, adCmdText
' Показать таблицу, если имеются записи
' в recordset
If Not rst.EOF Then
  ' Display the header
%>
  <TABLE BORDER>
  <TR>
    <TD><B>Firstname</B></TD>
    <TD><B>Lastname</B></TD>
    <TD><B>City</B></TD>
  </TR>

<%
  ' Выполнить цикл до
  ' вывода последей записи
  Do While Not rst.EOF
    ' Показать значения из таблицы
%>
```

```
    <TR>
      <TD><%= rst("FirstName") %></TD>
      <TD><%= rst("LastName") %></TD>
      <TD><%= rst("City") %></TD>
    </TR>
<%
    rst.MoveNext
  Loop
Else
%>
  <B>Нет записей в наборе!</B>
<%
End If
' Очистка
rst.Close
con.Close
%>
</TABLE>
</BODY>
</HTML>
```

На рис. 4.16 отображен экран при выполнении данного кода.

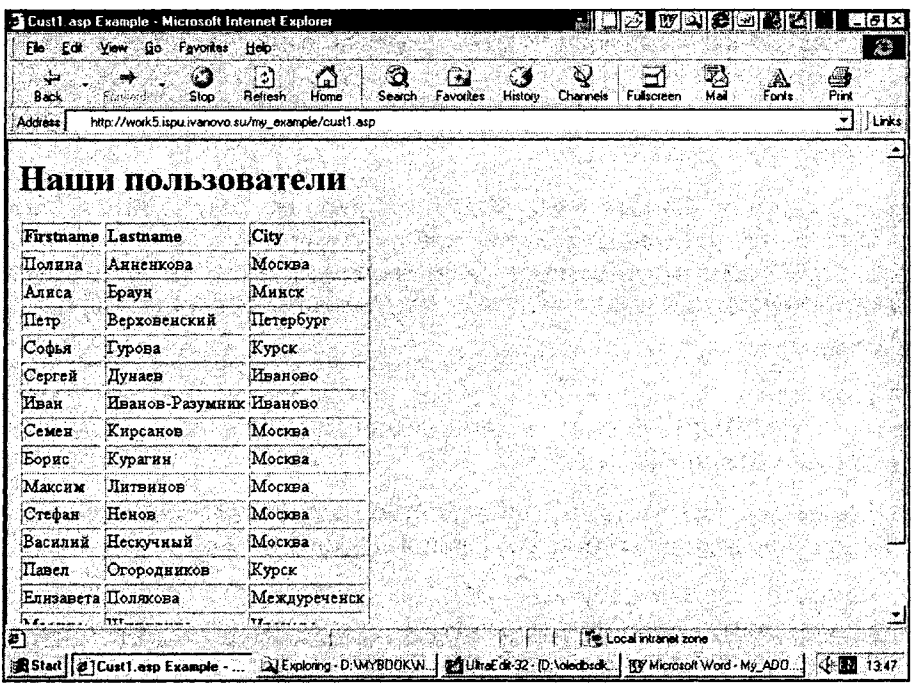


Рис. 4.16. Эта страница использует код ADO ASP, чтобы заполнить HTML-таблицу записями из таблицы Employees

Внимание!

Если вы хотите выполнить этот пример, то должны создать источник данных, называемый WebSQL, на вашей машине, которая выполняет сервер MS IIS Web и реализует запросы в базе данных WebSQL на сервере MS SQL. В следующих оставшихся разделах комментируются некоторые из частей исходного кода в cust1.asp.

Объявление переменных и констант. После необходимого HTML-заголовка, код VBScript в файле cust1.asp объявляет переменные и несколько констант, которые используются в методе `Open`:

```
<%
Dim con
Dim rst
Dim strSQL
Const adOpenStatic = 3
Const adLockReadOnly = 1
Const adCmdText = 1
```

Использование констант в вашем VBScript-коде вместо произвольных чисел — весьма привлекательная идея. Такая практика делает ваш код более читаемым.

Открытие соединения. Следующий код из файла cust1.asp создает объект `Connection` и открывает его:

```
Set con = Server.CreateObject("ADODB.Connection")
con.Open "WebSQL", "Employees", ""
```

Поскольку источник данных Web SQL указывает на базу данных SQL-сервера, вы должны обеспечить ввод пароля и пользовательского имени. Для баз данных Access эти параметры необязательные. Если вы используете Access вместо SQL-сервера, вы должны заменить последнюю строку на:

```
Con.Open "WebBase"
```

Первый параметр объекта `Connection` метод `Open`, `ConnectionString`, может обращаться или к ODBC имени источника данных (DSN), или к детализированной строке соединения ADO.

Создание `Recordset`. Эта часть кода создает набор записей (объект `recordset`), используя оператор SQL и связывая его с существующим объектом `Connection`:

```
strSQL = "SELECT * FROM Employees " & _
 "ORDER BY LastName, FirstName"
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open strSQL, con, adOpenStatic, _
 adLockReadOnly, adCmdText
```

Что, если записи отсутствуют? Cust1.asp включает последовательность операторов `If ... Then`, чтобы обработать случай, где запрос не возвращает никаких записей:

```
If Not rst.EOF Then
' ...создать таблицу ...
Else
%>
<B>Нет записей в наборе!</B>
<%
End If
```

Предшествующий код использует свойство `EOF`, чтобы проверить, присутствовали какие-либо записи. Если запрос возвращает пустой набор, ADO устанавливает `EOF`-свойство в значение "истина", в противном случае — в значение "ложь".

Показ таблицы. Если вы имеете записи в `recordset` (`rst.EOF = True`), тогда вы создаете заголовок таблицы при помощи следующих строк в исходном коде:

```
%>
<TABLE BORDER>
<TR>
 <TD><B>Firstname</B></TD>
 <TD><B>Lastname</B></TD>
 <TD><B>City</B></TD>
</TR>
```

После того как вы создали заголовок, необходимо организовать цикл для обработки поступающих записей и помещения их в HTML-таблицу. Это можно сделать при помощи цикла `Do ... While` и использования метода `MoveNext`.

```
<%
Do While Not rst.EOF
%>
    <TR>
      <TD><%= rst("FirstName") %></TD>
      <TD><%= rst("LastName") %></TD>
      <TD><%= rst("City") %></TD>
    </TR>
<%
    rst.MoveNext
  Loop
```

Модифицирование данных

При использовании ADO вы можете модифицировать записи, добавлять новые и удалять устаревшие записи. Следующие примеры показывают, как это можно сделать.

Добавление записей к Recordset. Чтобы добавить новую запись к набору данных ADO, используйте методы AddNew и Update объекта Recordset. Сначала вы должны создать объект Recordset. После того как вы создали объект Recordset, предназначенный не только для чтения, к нему можно добавить новую запись. Пример исходного кода:

```
rst.AddNew
rst("field1") = value1
rst("field2") = value2
' ...
rst.Update
```

Добавление новой записи — всегда процесс, состоящий из трех шагов:

1. Вызов метода AddNew, чтобы сообщить ADO, что вы хотите добавить новую пустую запись.
2. Установка значений одного или большего количества полей для новой записи.
3. Вызов метода Update, чтобы сохранить новую запись в базе данных.

Следующий пример состоит из двух файлов: HTML-файла, который фиксирует новую запись, используя форму HTML (addcust1.html), и ASP-файла (addcust1.asp), который добавляет новую запись в таблицу tblCustomers, используя методы Update и AddNew.

Содержимое файла addcust1.html:

```
<HTML>
<HEAD>
<TITLE>Addcust1.html Example</TITLE>
</HEAD>
<BODY bgcolor="wheat">
<H1> Добавление нового пользователя </H1>
<FORM METHOD="POST" ACTION="Addcust1.ASP">
<TABLE>
  <TR><TD>Имя:</TD>
      <TD><INPUT NAME="FirstName" SIZE=20></TD></TR>
  <TR><TD>Фамилия:</TD>
      <TD><INPUT NAME="LastName" SIZE=20></TD></TR>
  <TR><TD>Адрес:</TD>
      <TD><INPUT NAME="Address" SIZE=40></TD></TR>
  <TR><TD>Город:</TD>
      <TD><INPUT NAME="City" SIZE=20></TD></TR>
```

```
  <TR><TD>Область:</TD>
      <TD><INPUT NAME="State" SIZE=6></TD></TR>
  <TR><TD>Код:</TD>
      <TD><INPUT NAME="ZipCode" SIZE=10></TD></TR>
</TABLE>
<BR><INPUT TYPE="SUBMIT" VALUE="Сохранить запись">
<INPUT TYPE="RESET" VALUE="Отменить ввод">
<BR><BR>
</FORM>
</BODY>
</HTML>
```

Появляется страница Addcust1.html, показанная на рис. 4.17.

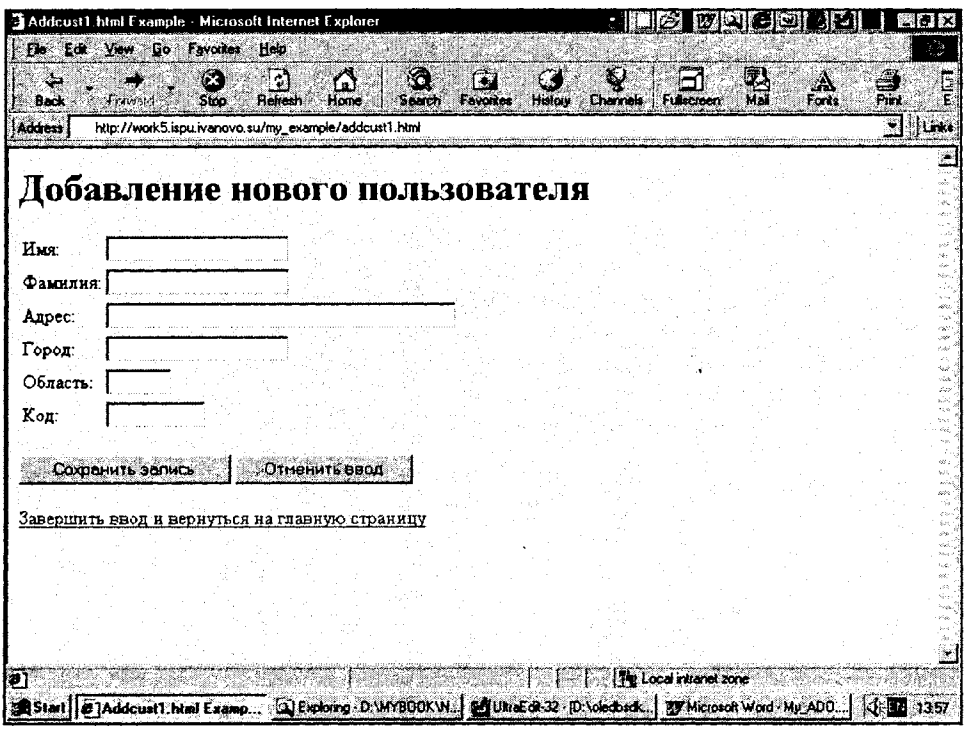


Рис. 4.17. Добавление новой записи в таблицу Employees

Файл addcust1.asp:

```
<HTML>
<HEAD>
<TITLE>Addcust1.asp Example</TITLE>
</HEAD>
```

```
<BODY>
<H1> Adding A New Customer </H1>
<%
Dim con
Dim rst
Const adOpenKeyset = 1
Const adLockOptimistic = 3
' Создать объекты
Set con = Server.CreateObject("ADODB.Connection")
Set rst = Server.CreateObject("ADODB.Recordset")
' Открыть соединение для Pubs базы данных
con.Open "WebSQL", "Employees", ""
' Создать keyset recordset, основаннный на
' Employees-таблице с оптимистической блокировкой
rst.Open "SELECT * FROM Employees", con, _
 adOpenKeyset, adLockOptimistic

' Создать пустую запись
rst.AddNew
  ' Установить поля для ввода пользовательских данных
  ' из html-формы
  rst("FirstName")= Request.Form("FirstName")
  rst("LastName")= Request.Form("LastName")
  rst("Address")= Request.Form("Address")
  rst("City")= Request.Form("City")
  rst("State")= Request.Form("State")
  rst("ZipCode")= Request.Form("ZipCode")
' Сохранить новую запись в базе данных
rst.Update
Response.Write "<B>Thank you!</B>"
rst.Close
con.Close
%>
<BR>
</BODY>
</HTML>
```

Поскольку файл addcust1.asp должен модифицировать данные, можно создать ключевой набор `recordset` с оптимистической блокировкой:

```
rst.Open "SELECT * FROM Employees", con, _
 adOpenKeyset, adLockOptimistic
```

Значения из вызываемой HTML-формы переносятся в код, использующий объект `Response`:

```
  rst("FirstName")= Request.Form("FirstName")
```

Когда новая запись успешно обработана, то появляется подтверждающая страница Addcust1.asp.

При желании всегда можно отредактировать любую запись из таблицы, написав код, подобный следующему:

```
<HTML>
<HEAD>
<TITLE>Update.asp Example</TITLE>
</HEAD>
<BODY bgcolor="lightblue">
<H1> Выбор пользователя для обновления</H1>
<%
' Объявить переменные
Dim con
Dim rst
Dim strSQL
' Объявить константы для Open-метода
Const adOpenStatic = 3
Const adLockReadOnly = 1
Const adCmdText = 1

' Создать Connection-объект и указатель
' к WebSQL источнику данных
Set con = Server.CreateObject("ADODB.Connection")
con.Open "WebSQL", "Employees", ""

' Создать Recordset-объект и заполнить его
' записями из SQL-оператора
strSQL = "SELECT EmployeeID, " & _
 "FirstName + ' ' + LastName As Name, " & _
 "City, Country " & _
 "FROM Employees " & _
 "ORDER BY LastName, FirstName"
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open strSQL, con, adOpenStatic, _
 adLockReadOnly, adCmdText

' Показать таблицу, если имеются записи
If Not rst.EOF Then
  ' Показать заголовок
%>
  <TABLE BORDER>
  <TR>
    <TD><B>Name</B></TD>
    <TD><B>City</B></TD>
    <TD><B>Country</B></TD>
  </TR>
```

```
<%
  ' Цикл по recordset
  Do While Not rst.EOF
    ' Показать значения записи
%>
    <TR>
      <!--Создать hyperlink в левой колонке,
          которая передает EmployeeId к
          UpdRec.asp-странице  -->
      <TD><A HREF="UpdRec.asp?EmployeeId=<%= rst("EmployeeId") %>">
          <%= rst("Name") %></A></TD>
      <TD><%= rst("City") %></TD>
      <TD><%= rst("Country") %></TD>
    </TR>
<%
    ' Перейти к следующей записи
    rst.MoveNext
  Loop

Else
%>
  <B>Нет записей в recordset!</B>
<%
End If

' Очистка
rst.Close
con.Close
%>
</TABLE>
<BR>
</BODY>
</HTML>
```

Эту страницу необходимо вызывать в браузере по URL, например:

http://work5/aspsamp/UpdRec.asp?EmployeeId=10

Страница, которая при этом появится в браузере, будет выглядеть подобно тому, как показано на рис. 4.18.

Если же вы не знаете точного идентификатора пользователя — можно написать следующую процедуру:

```
<HTML>
<HEAD>
<TITLE>UpdRec.asp Example</TITLE>
```

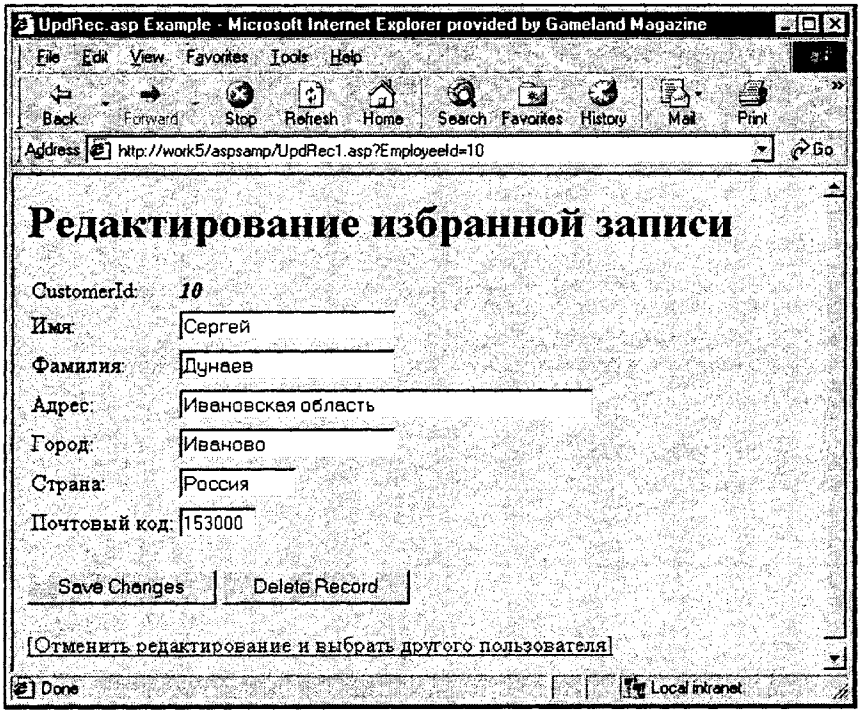


Рис. 4.18. Обновление записей в базе данных

```
</HEAD>
<BODY bgcolor="lightblue">
<H1> Редактирование записей </H1>
<%
' Объявление переменных
Dim con
Dim rst
Dim strSQL
' Объявление констант для метода Open
Const adOpenStatic = 3
Const adLockReadOnly = 1
Const adCmdText = 1

' Создание Connection объекта и указателя на
' WebSQL источник данных
Set con = Server.CreateObject("ADODB.Connection")
con.Open "WebSQL", "Employees", ""

' Создание recordset с указанной записью
' по номеру EmployeeId для
' процедуры update.asp
```

```
strSQL = "SELECT * FROM Employees " & _
 "WHERE EmployeeId =" & Request("EmployeeID")
Set rst = Server.CreateObject("ADODB.Recordset")
rst.Open strSQL, con, adOpenStatic, _
 adLockReadOnly, adCmdText

' Показываем форму с выбранной записью

If Not rst.EOF Then
%>
<FORM METHOD="POST" ACTION="UpdAct.ASP">
  <INPUT Type="HIDDEN" NAME="EmployeeID"
  Value="<% = rst("EmployeeId") %>">
<TABLE>

  <TR><TD>EmployeeId:</TD><TD>
  <B><I><% = rst("EmployeeId") %></I></B>
  </TD></TR>

  <TR><TD>First Name:</TD><TD>
  <INPUT NAME="FirstName"
  Size=20 Value="<% = rst("FirstName") %>">
  </TD></TR>

  <TR><TD>Last Name:</TD><TD>
  <INPUT NAME="LastName"
  Size =20 Value="<% = rst("LastName") %>">
  </TD></TR>

  <TR><TD>Address:</TD><TD>
  <INPUT NAME="Address"
  Size=40 Value="<% = rst("Address") %>">
  </TD></TR>

  <TR><TD>City:</TD><TD>
  <INPUT NAME="City"
  Size =20 Value="<% = rst("City") %>">
  </TD></TR>

  <TR><TD>PostalCode:</TD><TD>
  <INPUT NAME="PostalCode"
  Size =6 Value="<% = rst("PostalCode") %>">
  </TD></TR>

  <TR><TD>ZipCode:</TD><TD>
  <INPUT NAME="Country"
  Size =10 Value="<% = rst("Country") %>">
  </TD></TR>
```

```
</TABLE>
<BR><INPUT TYPE="SUBMIT" Name="Submit" VALUE="Save Changes">
    <INPUT TYPE="SUBMIT" Name="Submit" VALUE="Delete Record">
</FORM>

<%
Else
%>
  <B>Нет записи с номером данного EmployeeID!</B><BR>
<%
End If

' Очистка
rst.Close
con.Close
%>

<A HREF="update.asp">[Отменить редактирование]</A>
  

</BODY>
</HTML>
```

Тогда при подведении курсора к имени любого пользователя из списка, который отобразится в результате работы этой процедуры, будет автоматически вычисляться его идентификатор.

После корректировки записи (см. рис. 4.18) в процедуре `update.asp`, запись мгновенно изменяется, и вы можете убедиться в этом, запустив обслуживающую процедуру MS SQL Server и распечатав требуемую таблицу. В сущности, это является чрезвычайно удобным средством для удаленного управления таблицей на любом SQL-сервере, который поддерживает доступ через ODBC.

Ошибки обработки. В коде в update1.asp принимается, что новая запись всегда успешно добавляется к таблице `Employees`. К сожалению, вывод о том, что запись была добавлена именно в этой точке, — не является неоспоримым фактом. Некоторые вещи могут и будут делаться (время от времени) неправильно. Например, модификация может потерпеть неудачу по следующим причинам:

- нет прав на модификацию записи;
- нарушение целостности или индексная ошибка;
- наличие поля, которое не принимает пустые указатели — левый пробел;
- попытка записать строку в числовое поле;
- нарушение размерности поля.

Существует одно железное правило в программировании: если что-то, в принципе, может порождать ошибку, то эта ошибка будет порождаться всегда! (Особенно, когда вы находитесь в отпуске!)

Так что же случается, если вы сталкиваетесь с ошибкой при сохранении записи? Машина обработки сценариев ASP печатает ошибку на странице. Сообщение изменяется в зависимости от типа ошибки.

Обработка ошибок в VBScript. В отличие от своего кузена VBA, VBScript ограничил пользовательские возможности обработки ошибок. VBScript не поддерживает использование драйверов ошибки на уровне процедуры. Возможности обработки ошибок VBScript состоят из способности выключить заданную по умолчанию обработку ошибок (что позволяет вам перехватывать ошибки и обрабатывать их самостоятельно):

```
On Error Resume Next
```

Или возвращаться к заданной по умолчанию обработке ошибок:

```
On Error GoTo 0
```

Кроме того, когда вы выключаете заданную по умолчанию обработку ошибок, то можете использовать объект VBScript `Err`, чтобы получить информацию относительно ошибки. Например, вы можете проверить, произошла ли ошибка с оператором `If ... Then`, подобно следующим строкам кода:

```
If Err.Number <> 0 Then
' Может произойти ошибка
End If
```

Всякий раз, когда вы осуществляете проверку, значение `Err.Number` равняется `0`, если никакая ошибка не произошла, или числу (`Long Integer`), содержащему номер ошибки, которая произошла.

Вы также имеете доступ к сообщению об ошибках, которое показывает VBScript. Вы можете найти это сообщение, сохраненное в свойстве `Description` объекта `Err`, а также напечатать сообщение при помощи следующего оператора:

```
Response.Write Err.Description
```

Вы должны проверить значение `Err.Number` немедленно, потому что изменение этого значения случается после каждого оператора, и таким образом, вы будете располагать информацией относительно самой последней ошибки.

Совокупность ошибок ADO. В дополнение к объекту VBScript `Err`, объект `Connection` имеет набор ошибок, который может содержать многократные входы относительно последней ошибки ADO. Свойство совокупности ошибок `Count` сообщает вам и о других ошибках, если они находятся в наборе. Таким образом, с кодом, подобным нижеприведенному, вы сможете ото-

бражать все ошибки, которые могут произойти из операции ADO. (В следующем примере con указывает на активный объект ADO Connection.)

```
varErrorCount = con.Errors.Count
If varErrorCount > 0 Then
 For varI = 0 To varErrorCount - 1
  Response.Write "<BR><I>" & _
   con.Errors(varI).Description & "</I>"
 Next
End If
```

Так что же, все-таки, лучше использовать? Зачем нужна совокупность ошибок ADO, когда можно использовать более простой объект VBScript Err? Однако иногда специфическая операция ADO может генерировать ряд ошибок. В этих случаях, совокупность ошибок ADO обеспечивает дополнительную информацию относительно возникшей проблемы. Часто объект VBScript Err и совокупность ошибок ADO возвращает ту же самую информацию, так что вы со спокойной совестью, в большинстве случаев, можете использовать более простой объект Err. Тем не менее знать о том, что можно более подробно идентифицировать ошибки, все-таки нужно, и держать эти знания про запас. Независимо от того, какую форму идентификации ошибки вы выбрали, следует использовать оператор On Error Resume Next, чтобы сообщить VBScript, что вы сами хотите обрабатывать ошибки.

Пример обработки ошибок. Если вы решите выполнять пользовательскую обработку ошибок в ваших ASP-страницах, необходимо подготовить операцию включения внешнего кода. Поскольку в VBScript отсутствует способ определения процедурного уровня обработки ошибок, вы должны проверить, не случилась ли ошибка после каждого оператора VBScript или, по крайней мере, после каждого оператора VBScript, который может вызвать ошибку. Один из способов уменьшить нагрузку на систему при обработке ошибок состоит в том, чтобы создать функцию. Такая функция имеется в модификации к файлу addcust1.asp, именованному как addcust2.asp.

Функция HandleErrors заботится об обработке ошибок так, как показано в следующем коде:

```
<HTML>
<HEAD>
<TITLE>Addcust2.asp Example</TITLE>
</HEAD>
<BODY bgcolor="wheat">
<H1> Добавление нового пользователя </H1>
<%
Function HandleErrors(varNum, varDesc)
```

```
  ' Она показывает информацию об ошибках и
  ' возвращает  0, если нет ошибки, или
  '             1, если ошибка все-таки случилась
  On Error Resume Next
  If varNum <> 0 Then
    Response.Write "<BR><B>The following " & _
     "error occured:</B><BR>"
    Response.Write varDesc
    HandleErrors = 1
  Else
    HandleErrors = 0
  End If
End Function

On Error Resume Next
Dim con
Dim rst
Dim varErrors
Const adOpenKeyset = 1
Const adLockOptimistic = 3
varErrors = 0
' Создать объекты
Set con = Server.CreateObject("ADODB.Connection")
Set rst = Server.CreateObject("ADODB.Recordset")
' Открыть соединение с Pubs базой данных
con.Open "WebSQL", "Employees", ""
' Call HandleErrors after each operation that
' could generate an error.
varErrors = varErrors + _
 HandleErrors(Err.Number, Err.Description)
' Create a keyset recordset based on the
' Employees table with optimistic locking
rst.Open "SELECT * FROM Employees", con, _
 adOpenKeyset, adLockOptimistic
varErrors = varErrors + _
 HandleErrors(Err.Number, Err.Description)

' Создать пустую запись
rst.AddNew
varErrors = varErrors + _
 HandleErrors(Err.Number, Err.Description)

  ' Установить поля для ввода
  ' из html-формы
  rst("FirstName")= Request.Form("FirstName")
  rst("LastName")= Request.Form("LastName")
```

```
  rst("Address")= Request.Form("Address")
  rst("City")= Request.Form("City")
  rst("PostalCode")= Request.Form("PostalCode")
  rst("Country")= Request.Form("Country")
' Сохранить новую запись в базе данных
rst.Update
varErrors = varErrors + _
 HandleErrors(Err.Number, Err.Description)
Response.Write "<BR><BR>"
If varErrors = 0 Then
  Response.Write "<B>Запись добавлена!</B>"
Else
  Response.Write "<B>Запись не может быть добавлена " & _
   "поскольку имеется ошибка!</B>"
End If

rst.Close
con.Close
%>
<BR>
</BODY>
</HTML>
```

Таким образом, можно сказать, что программирование с использованием ADO представляет собой вовсе не трудную задачу. Особая ценность этого интерфейса состоит в том, что его, почти без всяких изменений, можно использовать где угодно — в приложениях MS Office, в программах на Visual Basic и Visual C++, в обыкновенных HTML- и ASP-файлах с прописанными сценариями. ADO опирается на OLE DB и в этом его главное достоинство, потому что последний стандарт является по-настоящему творческим произведением. Он подлинно универсален, всемирно признан и, главное, уже сейчас используется повсеместно, а не только в программных продуктах Microsoft.

JSP-сценарии

Что такое JavaServer Pages

Это новое мощное средство для придания Web-страницам гибкого, динамически обновляемого содержания — очень нужный инструмент для разработки систем электронной коммерции. JSP-функции в современном сервере приложений — это повторно используемый код, основанный на последней по времени появления спецификации JSP (JavaServer Pages Specification) фирмы Sun Microsystems.

Одна из главных особенностей JSP — это разделение бизнес-логики и представления данных на Web-страницах. JSP как раз и помогает реализовать некоторые алгоритмы, связанные с данными, представляемыми клиентам, поскольку они помогают осуществить доступ к повторно используемым компонентам, таким как сервлеты, JavaBeans и написанные на Java Web-приложения. JSP также поддерживает встроенный Java-код внутри Web-страниц.

Модели доступа к JSP

JSP-файлы могут быть доступны двумя способами:

- JSP-файл получает доступ к Bean-компонентам или другим компонентам, которые генерируют динамическое содержание страниц, посылаемых в браузер через JSP (рис. 4.19);

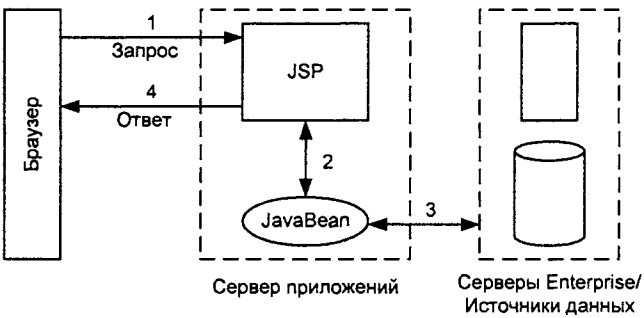


Рис. 4.19. Схема использования JSP по модели 1

- браузер посылает запрос к JSP-файлу (или непосредственно через JSP Engine, или через сервлет-посредник (рис. 4.20)).

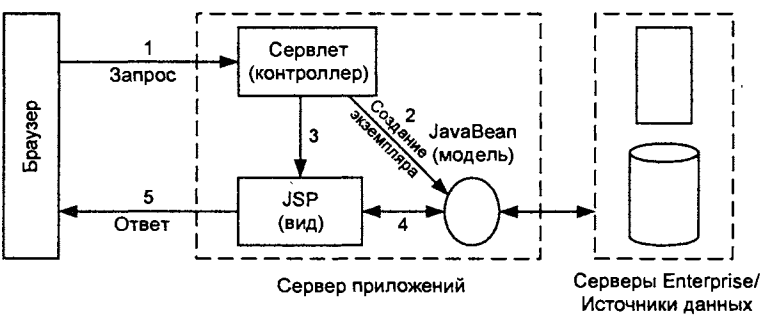


Рис. 4.20. Схема использования JSP по модели 2

Когда Web-сервер получает запрос к JSP-файлу, он переадресует его на сервер приложений, который может быть, а может и не быть его собственной

составной частью. Сервер приложений анализирует JSP-файл и определяет, какие в нем имеются теги, в случае необходимости генерируется Java-код, который компилируется и выполняется как сервлет. Генерация Java-кода происходит только при первом вызове сервлета, если оригинал JSP-файла не модифицировался. В последнем случае, сервер приложений обнаруживает изменение, регенерирует и компилирует сервлет заново перед его выполнением. Рис. 4.21 иллюстрирует этот процесс.

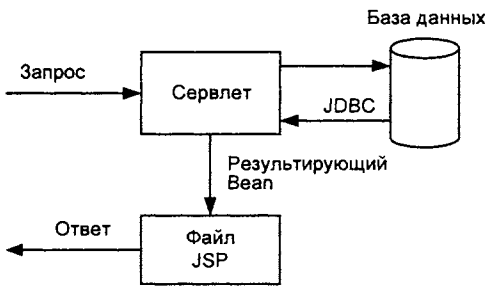


Рис. 4.21. Доступ к серверу

На сервере приложений при этом применяется набор новых методов в виде объекта `HttpServiceRequest` и объекта `HttpServiceResponse`. Эти методы позволяют вызываемому сервлету помещать объект (обычно Bean) в объектный запрос и передавать этот запрос, опять-таки, в страницу JSP для показа на дисплее. Вызываемая страница извлекает Bean из объекта запроса и опять использует JSP, но уже для того, чтобы сгенерировать HTML-код, понятный браузеру.

Имеются возможности для непосредственного вызова подготовленных JSP-страниц из браузера по их URL-адресам, но для этого на Web-сервере должна работать встроенная машина JSP Engine. Вы можете смело использовать JSP-сценарии в качестве обычных файлов, подобных ASP-файлам, если уверены, что ваш Web-сервер включает поддержку JSP. К таким серверам относятся сервер J2EE, Oracle Web Application Server, IBM WebSphere, Apache с поддержкой JSP в виде проекта Jakarta, сервер приложений Enhydra и множество других.

Пример использования JSP-файла (с упрощенным синтаксисом IBM WebSphere)

Примерный сервлет и вызываемый им JSP-файл иллюстрируют здесь возможную схему доступа. Сервлет, который называется PopulateBeanServlet, обеспечивает некую бизнес-логику. Его может разрабатывать программист, отвечающий за реализацию алгоритмов, связанных с обработкой данных. Сервлет вызывается из Web-страницы, содержащей форму вызова (HTML-

форма). Эту форму обеспечивает сотрудник, знакомый с проектированием страниц на HTML. Сервлет выполняет следующую работу:

- создает экземпляр DataBean (называемый `dataBean`);
- устанавливает Bean-параметры;
- устанавливает Bean (`dataBean`) как атрибут текущего объектного запроса;
- вызывает JSP-файл (DisplayData.jsp).

Тот же сотрудник, который оформляет HTML-страницы, может разработать и файл DisplayData.jsp. JSP-файл получает `dataBean`, который передается ему в объектном запросе и показывает свойства данного Bean.

Пример сервлета (PopulateBeanServlet)

```
import java.io.*;
import java.beans.Beans;

import javax.servlet.*;
import javax.servlet.http.*;

import DataBean;

/*****************************************************************
* PopulateBeanServlet — Этот сервлет создает экземпляр Bean
* (DataBean), устанавливает несколько его параметров, устанавливает Bean
* как атрибут в объектном запросе и вызывает JSP-файл для
* форматирования и показа данных, связанных с Bean
*****************************************************************/

public class PopulateBeanServlet extends HttpServlet
{

   public void Service(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
  {
      DataBean dataBean;

      // Создадим экземпляр объекта DataBean

      try
     {
         dataBean = (DataBean)
Beans.instantiate(this.getClass().getClassLoader(), "DataBean");
      }
      catch (Exception ex)
      {
         throw new ServletException("Не могу создать BEAN класса DataBean: "
                                    + ex.getMessage());
      }
```

```
      // Установим некоторые Bean-свойства
      dataBean.setProp1("Value1");
      dataBean.setProp2("Value2");
      dataBean.setProp3("Value3");

      // Чтобы послать Bean к JSP-файлу для форматирования и показа
      // содержания
      // 1) Установите Bean как атрибут в текущем объектном запросе
      ((com.sun.server.http.HttpServiceRequest)
        req).setAttribute("dataBean", dataBean);

      // 2) Используйте callPage, чтобы вызвать JSP-файл
      ((com.sun.server.http.HttpServiceResponse)
res).callPage("/DisplayData.jsp", req);

   }

} /* end of class PopulateBeanServlet */
```

Пример HTML-формы

```
<HTML>
<BODY>
<H1>Выполнение PopulateBeanServlet</H1>
<P>Желаете выполнить PopulateBeanServlet?
<FORM action="/servlet/PopulateBeanServlet" method="GET">
<INPUT type="SUBMIT" value="Yes">
<INPUT type="SUBMIT" value="No">
</FORM>
</BODY>
</HTML>
```

Пример результирующей JSP-страницы (DisplayData.jsp)

```
<!-Этот JSP-файл получает Bean, передаваемый ему в объектном запросе,
      и показывает Bean-свойства -->

<html>
<head>
<title>Bean Data Display</title>
</head>

<!-- Определим Bean, используя тег BEAN   -->
<bean name="dataBean" type="DataBean" introspect="no" create="no"
scope="request">
</bean>
<body>
```

```
<!—Имеется 3 способа, чтобы получить доступ к Bean-свойствам -->
<!--   1. Использование JSP-скриптлета  -->
<% out.println("Значение свойства 1 нашего Bean будет — " +
dataBeans.getProp1());
%>

<!--   2. Использование JSP-выражения -->
<p>Значение свойства 2 нашего Bean будет -
<%= dataBean.getProp2()  %> </p>

<--    3. Использование тега INSERT   -->
<p>Значение свойства 3 нашего Bean будет -
<insert bean=dataBean property=prop3 default="No property value" >
</insert></p>

</body>
</html>
```

Что могут содержать JSP-файлы

JSP-файлы имеют расширение .jsp. JSP-файлы могут содержать комбинацию HTML-тегов, NCSA-тегов (специальные теги, которые были первыми при использовании на Web-серверах некоторых программных включений), <SERVLET>-тегов (только для некотых типов JSP Engines) и JSP-синтаксиса.

HTML-теги

JSP поддерживает все действительные теги HTML. Вы можете обратиться к любому руководству по HTML.

Тег <SERVLET> (этот тег не входит в стандартную спецификацию Sun JSP 1.0).

Использование тега <SERVLET> — это метод встраивания сервлетов внутрь JSP-файла. Синтаксис тега следующий:

```
<SERVLET NAME="servlet_name" CODE="servlet_class_name"
CODEBASE="URL_for_remote_loading"
   initparm1="initparm_value">
<PARAM NAME="param_name" VALUE="param_value">
</SERVLET>
```

Теги NCSA (эти теги не входят в стандартную спецификацию Sun JSP 1.0).

В некоторых старых SHTML-файлах могут встречаться NCSA-теги (вместо сервлетов), которые применялись для сервер-ориентированных программных включений. Если ваш сервер приложений поддерживает NCSA-теги в SHTML-файлах, то можно преобразовать их в JSP-файлы и оставить в них

теги в формате NCSA. Например, WebSphere Application Server поддерживает следующие NCSA-теги, которые можно использовать в JSP:

- `config`
- `echo var=variable`
- `exec`
- `filesize`
- `include`
- `lastmodified`
- Команды форматирования и вывода данных

Общепринятый JSP-синтаксис (JSP-спецификация)

JSP-синтаксис образует следующие форматы:

- **Директивы**

  Используя JSP-директивы (заключенные между `<%@` и `%>`) можно определить:

  - Язык скрипта
  - Интерфейсы для сервлетов
  - Классы для расширения сервлета
  - Импортируемые пакеты

- **Объявление переменных и методов**
- **Встроенный Java-код (scriptlets), заключенный внутри `<%` и `%>`**

Вы можете встраивать в JSP-сценарий операторы языка Java. Такой встроенный код называется *скриптлет* (scriptlet). Эти скриптлеты, естественно, отличаются от тех, которые мы рассматривали выше в интерпретации фирмы Microsoft, но суть их, в принципе, та же — обеспечить дополнительную функциональность на сервере при интерпретации Web-страниц.

Сводная таблица (табл. 4.5) содержит описание синтаксических конструкций согласно спецификации JSP 1.0 Revision B.

Таблица 4.5. *Синтаксические конструкции в соответствии со спецификацией JSP 1.0 Revision B*

Назначение тега	Описание. JSP-синтаксис
Выводные строки комментариев	Посылаются к клиенту для просмотра `<!- — comment [ <%= expression %> ] -->`

Таблица 4.5 *(продолжение)*

Назначение тега	**Описание. JSP-синтаксис**
Скрытые комментарии	Не посылаются клиенту, а используются для внутреннего документирования `<%- — comment --%>`
Объявления	Объявления переменных и методов, действительных для языка сценариев `<%! declarations %>`
Выражения	Действительные допустимые выражения `<%= expression %>`
Скриптлет	Содержит кодовые фрагменты на языке сценариев (как правило — Java) `<% code fragment %>`
Include-директива	Включает текст или код в JSP-файл `<%@ include file=" relativeURL "%>`

Page-директива	Определяет некоторые атрибуты, применимые по отношению к JSP-странице
	`<%@ page [ language=" java " ] [ extends=" package. class " ] [ import= " { package. class | package .* }, ... " ]`
	`[ session=" true |false" ] [ buffer=" none| 8kb | size kb" ] [ autoFlush=" true |false" ]`
	`[ isThreadSafe=" true |false" ] [ info=" text " ] [ errorPage=" relativeURL "]`
	`[ contentType=" mimeType [; charset =characterSet ]"|" text/ html; charset= ISO- 8859- 1 "]`
	`[ isErrorPage=" true| false "] %>`

TagLib-директива	Определяет библиотеки и префикс для пользовательских тегов, вставляемых в JSP-страницы
	`<%@ taglib uri=" URIToTagLibrary " prefix=" tagPrefix "%>`
	Пользовательский тег:
	`< tagPrefix : name attribute =" value "+ ... />`
	`< tagPrefix : name attribute =" value "+ ... > other tags </ tagPrefix : name >`

`<jsp:forward>`	Передает клиентский запрос к HTML-файлу, JSP-файлу или сервлету
	`<jsp: forward page="{ relativeURL |`
	`<%= expression %> }" />`

Таблица 4.5 *(продолжение)*

Назначение тега	Описание. JSP-синтаксис
<jsp:getProperty>	Получает свойства Bean-компонента, так чтобы их можно было показать на дисплее <jsp: getProperty name=" beanInstanceName " property=" propertyName "/>
<jsp:include>	Включает данные в JSP-страницу из другого файла без всякого их анализа <jsp: include page="{ relativeURL | <%= expression %> }" flush=" true" />
<jsp:plugin>	Загружает Java-plugin на клиентский браузер, чтобы выполнить в нем апплет или Bean <jsp: plugin type=" bean| applet" code=" classFileName " codebase=" classFileDirectoryName " [name=" instanceName "] [archive=" URIToArchive, ... "] [align=" bottom |top| middle| left| right"] [height=" displayPixels "] [width=" displayPixels "] [hspace=" leftRightPixels "] [vspace=" topBottomPixels "] [jreversion=" JREVersionNumber | 1.1 "] [nspluginurl=" URLToPlugin "] [iepluginurl=" URLToPlugin "] > [<jsp: params> [<jsp: param name=" parameterName " value=" parameterValue " />]+ </ jsp: params>] [<jsp: fallback> text message for user </ jsp: fallback>] </ jsp: plugin>
<jsp:setProperty>	Устанавливает свойства Bean-компонента <jsp: setProperty name=" beanInstanceName " { property= "*"| property=" propertyName " [param=" parameterName "]| property=" propertyName " value=" { string | <%= expression %>}"}/>
<jsp:useBean>	Размещает или инициирует Bean-компонент с собственным именем и контекстной областью (scope) <jsp: useBean id=" beanInstanceName " scope=" page |request| session| application" { class=" package. class " | type=" package. class " | class=" package. class " type=" package. class "| beanName=" { package. class | <%= expression %> } " type=" package. class "} {/>| > other tags </ jsp: useBean> }

Таблица 4.5 (окончание)

Подразумеваемые объекты:	Тип	Scope	Некоторые полезные методы
request	*javax. servlet. ServletRequest*	Request	getParameter, getParameterNames, getParameterValues
response	*javax. servlet. ServletResponse*	Page	обычно не используется
pageContext	*javax. servlet. jsp. PageContext*	Page	findAttribute, getAttribute, getAttributesScope, getAttributeNamesInScope
session	*javax. servlet. http. HttpSession*	Session	getId, getValue, getValueNames, putValue
application	*javax. servlet. ServletContext*	Application	getMimeType, getRealPath
out	*javax. servlet. jsp. JspWriter*	Page	clear, clearBuffer, flush, getBufferSize, getRemaining
config	*javax. servlet. ServletConfig*	Page	getInitParameter, getInitParameterNames
page	*java. lang. Object*	Page	обычно не используется
exeption	*java. lang. Throwable*	Page	getMessage, getLocalizedMessage, printStackTrace

JSP-API и примеры стандартных JSP-сценариев

Все интерфейсы, связанные с JSP-технологией, изложены в основополагающем документе JSP-спецификации, которая входит в состав дистрибутивного файла JSWDK. При взаимодействии сервлетов и JSP-файлов используются в основном два главных интерфейса:

```
javax.servlet.http.HttpServletRequest
com.sun.server.http.HttpServiceResponse
```

Следующие примеры хорошо иллюстрируют процесс обмена.

1. Если вы используете jswdk 1.0.1, то сначала выполните:

```
cd \jswdk-1.0.1
set JAVA_HOME-d:\jdk1.2
startserver.bat
```

Затем:

```
http://127.0.0.1:8080/examples/jsp/jsptoserv/jsptoservlet.jsp
```

2. Распечатка файла jsptoservlet.jsp.

```
<html>
<!--
  Пример первичного вызывающего сервлет JSP-файла
  Вызывается с любого сервера приложений по URL вида:
  http://127.0.0.1:8080/examples/jsp/jsptoserv/jsptoservlet.jsp
  Если вы используете jswdk-1.0.1, то сначала выполните:
  set JAVA_HOME-d:\jdk1.2
  startserver.bat
-->
<body bgcolor="white">

<!-- Переадресовать к сервлету -->
<jsp:forward page="/servlet/servletToJsp" />

</html>
```

3. Распечатка исходного кода программы сервлета servletToJsp.java.

```
import javax.servlet.*;
import javax.servlet.http.*;

public class servletToJsp extends HttpServlet {

    public void doGet (HttpServletRequest request,
                       HttpServletResponse response) {

    try {
        // Установка атрибутов и переадресовка к hello.jsp
        request.setAttribute ("servletName", "servletToJsp");
getServletConfig().getServletContext().getRequestDispatcher
("/jsp/jsptoserv/hello.jsp").forward(request, response);
    } catch (Exception ex) {
        ex.printStackTrace ();
    }
    }
}
```

4. Распечатка файла hello.jsp.

```
<html>
<!--
  Этот JSP-сценарий вызывается непосредственно
  из сервлета, имя которого передается ему как параметр
-->
<body bgcolor="magenta">
```

```
<h1>
Я – Hello.jsp – был вызван при помощи
<% out.print (request.getAttribute("servletName").toString()); %>
 сервлета.
</h1>

</html>
        getServletCon-
fig().getServletContext().getRequestDispatcher("/jsp/jsptoserv
/hello.jsp").forward(request, response);
    } catch (Exception ex) {
        ex.printStackTrace ();
    }
    }
}
```

Результат выполнения сценария показан на рис. 4.22.

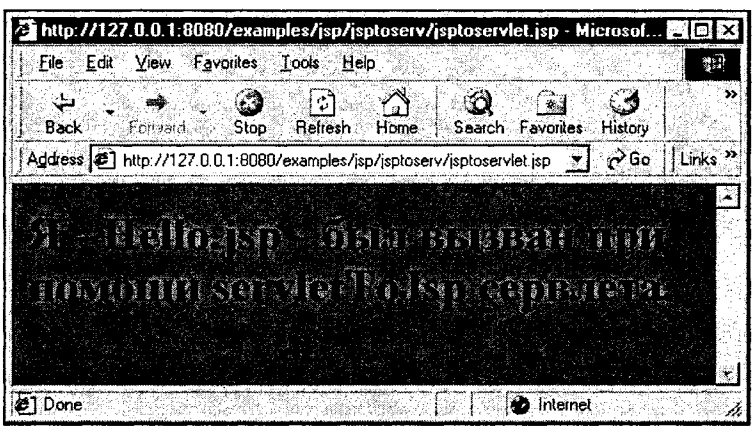


Рис. 4.22. Выполнение файла JSP во взаимодействии с сервлетом

Средства для создания JSP-файлов

В настоящее время существует большое многообразие различных программных средств для автоматизированного проектирования JSP-страниц. Лучшие из них перечислены ниже.

IBM WebSphere Studio Version 3.0

Это прекрасное программное средство, работа с которым доставляет истинное удовольствие. Пакет снабжен мастерами, графическими редакторами, можно схематично отображать все взаимосвязи в узле, проектировать и подключать сервлеты, JavaBean-компоненты и JSP-сценарии, добиваться изощ-

ренного дизайна для отображения живого содержания Web-страниц. Кроме того, этот пакет, как нельзя лучше подходит для работы с базами данными, особенно с DB2. Мастера включают поддержку SQL-запросов к реляционным базам данных и обеспечивают комплекс информационных услуг для накопления информации о Web-визитерах. Специальный Studio Wizard создает сервлеты, JavaBeans и JSP для проектирования динамически обновляемого контента. Великолепные графические средства превращают рабочее место в настоящую Web-студию художника, снабженную всеми необходимыми средствами. Более подробную информацию об этом пакете можно найти на сайте IBM:

http://www.software.ibm.com/webservers/studio/index.html.

NetObjects ScriptBuilder

JSP-поддержка в ScriptBuilder включает готовые шаблоны JSP, многочисленные примеры программирования, которые можно взять за основу, и полную поддержку стандартного JSP-синтаксиса. Более подробная информация находится на сайте

http://www.netobjects.com/.

IBM VisualAge for Java Enterprise 3.0

VisualAge for Java 3.0 обеспечивает следующие средства для разработки JSP:

- Servlet Launcher — позволяет стартовать Web-сервер, открывает ваш Web-браузер и запускает сервлет;
- JSP-монитор выполнения — позволяет отслеживать процесс выполнения JSP, генерировать сервлеты и HTML-коды;
- VisualAge for Java также позволяет устанавливать точки прерывания внутри сервлет-кодов, динамически их изменять и пошагово исполнять процедуры (**http://www.software.ibm.com/ad/vajava/**).

Главная цель изобретения JSP

Технология JSP была создана для совершенно другой цели, нежели быть простым и удобным заменителем сервлетов. Нет, наоборот, сервлеты являются неотъемлемой частью JSP, а главная цель изобретения JSP — это разделение дизайна проектируемых страниц и программной части, формирующей их динамическое содержание. Несмотря на то, что вы можете вставить любое количество Java-кодов в страницу JSP, все-таки лучше включить бизнес-логику в компоненты JavaBean. Идея очень проста — каждый должен заниматься своим делом: художник проектировать внешний дизайн, а программист создавать инструменты для его наполнения живым содержанием, т. е. информацией. Причем программы ни в коей мере не должны зависеть

от дизайна, а дизайн от программ. Одна из основных особенностей программирования JSP заключается в делегировании функций обработки компонетам JavaBean. Благодаря этому можно с удивительной легкостью манипулировать JavaBean-компонентами на Web-страницах и по своему усмотрению менять и динамически профилировать информационный контекст. JSP обладают такими возможностями, каких еще никогда не существовало.

Использование JavaBean-компонентов в JSP

Поскольку компоненты JavaBean являются коммерчески доступными (иногда и совершенно бесплатными), повторно используемыми, универсальными компонентами, которые можно приобрести, скачать, разработать и использовать где угодно, в любом программном средстве на любой платформе, естественно, возникает искушение использовать их в JSP и поручать им реализацию бизнес-алгоритмов, так, чтобы потом, с максимальной степенью комфорта, воспользоваться этими результатами. Удобство заключается еще и в том, что вы работаете в распределенной среде, которая полностью для вас прозрачна — у вас, к примеру, только Web-браузер, а сверхмощный сервер приложений с комплектом JSP-страниц и связанных с ними Bean-компонентами находится где-то далеко. Для этих целей, т. е. включения Bean-компонентов в JSP-страницы, активизации их и передачи им параметров, существует специальный тег:

```
<jsp:useBean>
```

Рассмотрим конкретный пример использования этого тега при работе с календарем, оформленным в виде Bean-программы. Чтобы выполнить у себя на компьютере эти примеры, можно использовать стандартную бесплатную поставку фирмы Sun для использования JSP-технологии — пакет программ JSWDK.

Для серверов приложений на основе JSWDK

1. Распечатка файла date.jsp.

```
<html>
<!--
  *Пример из стандартного дистрибутива JSWDK

-->

<body bgcolor="white">
<jsp:useBean id='clock' scope='page' class='dates.JspCalendar'
type="dates.JspCalendar" />

<font size=4>
<ul>
```

```
<li>   Day of month: is  <jsp:getProperty name="clock" prop-
erty="dayOfMonth"/>
<li>   Month: is  <jsp:getProperty name="clock" property="month"/>
<li>   Year: is  <jsp:getProperty name="clock" property="year"/>
<li>   Time: is  <jsp:getProperty name="clock" property="time"/>
<li>   Date: is  <jsp:getProperty name="clock" property="date"/>
<li>   Day: is  <jsp:getProperty name="clock" property="day"/>
<li>   Day Of Year: is  <jsp:getProperty name="clock" prop-
erty="dayOfYear"/>
<li>   Week Of Year: is  <jsp:getProperty name="clock" prop-
erty="weekOfYear"/>
<li>   era: is  <jsp:getProperty name="clock" property="era"/>
<li>   DST Offset: is  <jsp:getProperty name="clock" prop-
erty="dSTOffset"/>
<li>   Zone Offset: is  <jsp:getProperty name="clock" prop-
erty="zoneOffset"/>
</ul>
</font>

</body>
</html>
```

2. Распечатка файла jspCalendar.java.

```
/*
 * Пример из стандартного дистрибутива JSWDK
 *
 */

package dates;

import java.text.DateFormat;
import java.util.*;

public class JspCalendar {
    Calendar  calendar = null;

    public JspCalendar() {
        calendar = Calendar.getInstance(TimeZone.getTimeZone("PST"));
        Date trialTime = new Date();
        calendar.setTime(trialTime);
    }

    public int getYear() {
        return calendar.get(Calendar.YEAR);
    }

    public String getMonth() {
        int m = getMonthInt();
```

```
    String[] months = new String [] { "January", "February", "March",
                                      "April", "May", "June",
                                      "July", "August", "September",
                                      "October", "November", "December" };
    if (m > 12)
        return "Unknown to Man";

    return months[m - 1];

}

public String getDay() {
    int x = getDayOfWeek();
    String[] days = new String[] {"Sunday", "Monday", "Tuesday",
                                   "Thursday", "Friday", "Saturday"};
                                   "Wednesday",

    if (x > 7)
        return "Unknown to Man";

    return days[x - 1];

}

public int getMonthInt() {
    return 1 + calendar.get(Calendar.MONTH);
}

public String getDate() {
    return getMonthInt() + "/" + getDayOfMonth() + "/" +  getYear();

}

public String getTime() {
    return getHour() + ":" + getMinute() + ":" + getSecond();
}

public int getDayOfMonth() {
    return calendar.get(Calendar.DAY_OF_MONTH);
}

public int getDayOfYear() {
    return calendar.get(Calendar.DAY_OF_YEAR);
}

public int getWeekOfYear() {
    return calendar.get(Calendar.WEEK_OF_YEAR);
}
```

```
public int getWeekOfMonth() {
   return calendar.get(Calendar.WEEK_OF_MONTH);
}

public int getDayOfWeek() {
   return calendar.get(Calendar.DAY_OF_WEEK);
}

public int getHour() {
   return calendar.get(Calendar.HOUR_OF_DAY);
}

public int getMinute() {
   return calendar.get(Calendar.MINUTE);
}


public int getSecond() {
   return calendar.get(Calendar.SECOND);
}

public static void main(String args[]) {
   JspCalendar db = new JspCalendar();
   p("date: " + db.getDayOfMonth());
   p("year: " + db.getYear());
   p("month: " + db.getMonth());
   p("time: " + db.getTime());
   p("date: " + db.getDate());
   p("Day: " + db.getDay());
   p("DayOfYear: " + db.getDayOfYear());
   p("WeekOfYear: " + db.getWeekOfYear());
   p("era: " + db.getEra());
   p("ampm: " + db.getAMPM());
   p("DST: " + db.getDSTOffset());
   p("ZONE Offset: " + db.getZoneOffset());
    //     p("TIMEZONE: " + db.getUSTimeZone());
}

private static void p(String x) {
   System.out.println(x);
}


public int getEra() {
   return calendar.get(Calendar.ERA);
}

public String getUSTimeZone() {
   String[] zones = new String[] {"Hawaii", "Alaskan", "Pacific",
                                  "Mountain", "Central", "Eastern"};
```

```
        int index = 10 + getZoneOffset();

        if (index <= 5) {
            return zones[10 + getZoneOffset()];
        }
        else return "Only US Time Zones supported";
    }

    public int getZoneOffset() {
        return calendar.get(Calendar.ZONE_OFFSET)/(60*60*1000);
    }


    public int getDSTOffset() {
        return calendar.get(Calendar.DST_OFFSET)/(60*60*1000);
    }


    public int getAMPM() {
        return calendar.get(Calendar.AM_PM);
    }
}
```

В этом примере все предельно понятно, нет даже смысла его подробно объяснять, учитывая все вышеизложенное. После компиляции и построения JavaBean-компонента этот пример можно запустить в браузере.

Сначала выполните:

```
 cd \jswdk-1.0.1
set JAVA_HOME=d:\jdk1.2
 startserver.bat
```

Затем из браузера:

```
http://127.0.0.1:8080/examples/jsp/dates/date.jsp
```

Результат показан на рис. 4.23.

Точно таким же способом можно выполнить и другие примеры из состава JSWDK, каталог /JSWDK-1.0.1/examples.

Хотя значительная часть программистов работает со стандартным синтаксисом JSP-страниц, поддерживаемым большинством фирм, все-таки существуют некоторые нюансы использования коммерчески доступных серверов приложений, среди которых первое место, несомненно, принадлежит известному продукту *IBM WebSphere*, в который недавно, по сообщениям печати, компания IBM инвестировала 1 миллиард долларов!!! Судя по всему, это наиболее впечатляющая инвестиция в отдельный программный продукт, который объединяется с такой известной системой, также ныне входящей в круг компетенции IBM, — *Lotus Domino Server*. IBM становится наиболее

крупным игроком на рынке создания средств для разработки Web-приложений, причем ориентированных на самые перспективные и авангардные технологии. Так вот, при использовании JavaBeans с WebSphere существуют некоторые отличия. Кратко остановимся на них.

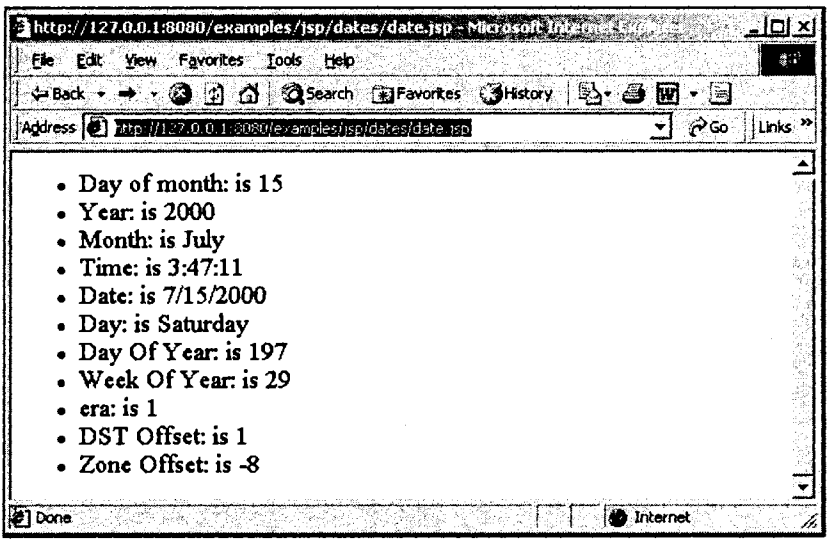


Рис. 4.23. Сценарий JSP с применением готового компонента JavaBean

Использование апплетов в JSP (plugin)

В этом примере демонстрируется возможность встраивания апплетов в страницы JSP, которые выполняются на сервере.

1. Распечатка plugin.jsp.

```
<html>
<!-- Copyright (c) 1999 Sun Microsystems, Inc. All Rights Reserved.
-->
<title> Plugin example </title>
<body bgcolor="white">
<h3> Current time is : </h3>
<jsp:plugin type="applet" code="Clock2.class" code-
base="/examples/jsp/plugin/applet" jreversion="1.2" width="160"
height="150" >
    <jsp:fallback>
        Plugin tag OBJECT or EMBED not supported by browser.
    </jsp:fallback>
</jsp:plugin>
<p>
```

```
<h4>
<font color=red>
The above applet is loaded using the Java Plugin from a jsp page
using the
plugin tag.
</font>
</h4>
</body>
</html>
```

2. Распечатка текста апплета clock2.java.

```
/*
 * @(#)Clock2.java    1.5 98/07/13

 */

import java.util.*;
import java.awt.*;
import java.applet.*;
import java.text.*;

/**
 * Time!
 *
 * @author Rachel Gollub
 */

public class Clock2 extends Applet implements Runnable {
    Thread timer;                  // The thread that displays clock
    int lastxs, lastys, lastxm,
        lastym, lastxh, lastyh;    // Dimensions used to draw hands
    SimpleDateFormat formatter;    // Formats the date displayed
    String lastdate;               // String to hold date displayed
    Font clockFaceFont;            // Font for number display on clock
    Date currentDate;              // Used to get date to display
    Color handColor;               // Color of main hands and dial
    Color numberColor;             // Color of second hand and numbers

    public void init() {
        int x,y;
        lastxs = lastys = lastxm = lastym = lastxh = lastyh = 0;
        formatter = new SimpleDateFormat ("EEE MMM dd hh:mm:ss yyyy",
Locale.getDefault());
        currentDate = new Date();
        lastdate = formatter.format(currentDate);
        clockFaceFont = new Font("Serif", Font.PLAIN, 14);
        handColor = Color.blue;
        numberColor = Color.darkGray;
```

```
        try {
            setBackground(new
Color(Integer.parseInt(getParameter("bgcolor"),16)));
        } catch (Exception E) { }
        try {
            handColor = new
Color(Integer.parseInt(getParameter("fgcolorl"),16));
        } catch (Exception E) { }
        try {
            numberColor = new
Color(Integer.parseInt(getParameter("fgcolor2"),16));
        } catch (Exception E) { }
        resize(300,300);              // Set clock window size
    }

    // Plotpoints allows calculation to only cover 45 degrees of the circle,
    // and then mirror
    public void plotpoints(int x0, int y0, int x, int y, Graphics g) {
        g.drawLine(x0+x,y0+y,x0+x,y0+y);
        g.drawLine(x0+y,y0+x,x0+y,y0+x);
        g.drawLine(x0+y,y0-x,x0+y,y0-x);
        g.drawLine(x0+x,y0-y,x0+x,y0-y);
        g.drawLine(x0-x,y0-y,x0-x,y0-y);
        g.drawLine(x0-y,y0-x,x0-y,y0-x);
        g.drawLine(x0-y,y0+x,x0-y,y0+x);
        g.drawLine(x0-x,y0+y,x0-x,y0+y);
    }

    // Circle is just Bresenham's algorithm for a scan converted circle
    public void circle(int x0, int y0, int r, Graphics g) {
        int x,y;
        float d;
        x=0;
        y=r;
        d=5/4-r;
        plotpoints(x0,y0,x,y,g);

        while (y>x){
            if (d<0) {
                d=d+2*x+3;
                x++;
            }
            else {
                d=d+2*(x-y)+5;
                x++;
                y--;
            }
```

```
            plotpoints(x0,y0,x,y,g);
        }
    }

    // Paint is the main part of the program
    public void paint(Graphics g) {
        int xh, yh, xm, ym, xs, ys, s = 0, m = 10, h = 10, xcenter, ycenter;
        String today;

        currentDate = new Date();
        SimpleDateFormat formatter = new
                                SimpleDateFormat("s",Locale.getDefault());
        try {
            s = Integer.parseInt(formatter.format(currentDate));
        } catch (NumberFormatException n) {
            s = 0;
        }
        formatter.applyPattern("m");
        try {
            m = Integer.parseInt(formatter.format(currentDate));
        } catch (NumberFormatException n) {
            m = 10;
        }
        formatter.applyPattern("h");
        try {
            h = Integer.parseInt(formatter.format(currentDate));
        } catch (NumberFormatException n) {
            h = 10;
        }
        formatter.applyPattern("EEE MMM dd HH:mm:ss yyyy");
        today = formatter.format(currentDate);
        xcenter=80;
        ycenter=55;

    // a= s* pi/2 - pi/2 (to switch 0,0 from 3:00 to 12:00)
    // x = r(cos a) + xcenter, y = r(sin a) + ycenter

        xs = (int)(Math.cos(s * 3.14f/30 - 3.14f/2) * 45 + xcenter);
        ys = (int)(Math.sin(s * 3.14f/30 - 3.14f/2) * 45 + ycenter);
        xm = (int)(Math.cos(m * 3.14f/30 - 3.14f/2) * 40 + xcenter);
        ym = (int)(Math.sin(m * 3.14f/30 - 3.14f/2) * 40 + ycenter);
        xh = (int)(Math.cos((h*30 + m/2) * 3.14f/180 - 3.14f/2) * 30
                + xcenter);
        yh = (int)(Math.sin((h*30 + m/2) * 3.14f/180 - 3.14f/2) * 30 +
                ycenter);

    // Draw the circle and numbers

        g.setFont(clockFaceFont);
        g.setColor(handColor);
```

```
        circle(xcenter,ycenter,50,g);
        g.setColor(numberColor);
        g.drawString("9",xcenter-45,ycenter+3);
        g.drawString("3",xcenter+40,ycenter+3);
        g.drawString("12",xcenter-5,ycenter-37);
        g.drawString("6",xcenter-3,ycenter+45);

    // Erase if necessary, and redraw

        g.setColor(getBackground());
        if (xs != lastxs || ys != lastys) {
            g.drawLine(xcenter, ycenter, lastxs, lastys);
            g.drawString(lastdate, 5, 125);
        }
        if (xm != lastxm || ym != lastym) {
            g.drawLine(xcenter, ycenter-1, lastxm, lastym);
            g.drawLine(xcenter-1, ycenter, lastxm, lastym); }
        if (xh != lastxh || yh != lastyh) {
            g.drawLine(xcenter, ycenter-1, lastxh, lastyh);
            g.drawLine(xcenter-1, ycenter, lastxh, lastyh); }
        g.setColor(numberColor);
        g.drawString("", 5, 125);
        g.drawString(today, 5, 125);
        g.drawLine(xcenter, ycenter, xs, ys);
        g.setColor(handColor);
        g.drawLine(xcenter, ycenter-1, xm, ym);
        g.drawLine(xcenter-1, ycenter, xm, ym);
        g.drawLine(xcenter, ycenter-1, xh, yh);
        g.drawLine(xcenter-1, ycenter, xh, yh);
        lastxs=xs; lastys=ys;
        lastxm=xm; lastym=ym;
        lastxh=xh; lastyh=yh;
        lastdate = today;
        currentDate=null;
    }

    public void start() {
        timer = new Thread(this);
        timer.start();
    }

    public void stop() {
        timer = null;
    }

    public void run() {
        Thread me = Thread.currentThread();
```

```
        while (timer == me) {
            try {
                Thread.currentThread().sleep(100);
            } catch (InterruptedException e) {
            }
            repaint();
        }
    }

    public void update(Graphics g) {
        paint(g);
    }

    public String getAppletInfo() {
        return "Title: A Clock \nAuthor: Rachel Gollub, 1995 \nAn analog
clock.";
    }

    public String[][] getParameterInfo() {
        String[][] info = {
            {"bgcolor", "hexadecimal RGB number", "The background color.
Default is the color of your browser."},
            {"fgcolor1", "hexadecimal RGB number", "The color of the
hands and dial. Default is blue."},
            {"fgcolor2", "hexadecimal RGB number", "The color of the
seconds hand and numbers. Default is dark gray."}
        };
        return info;
    }
}
```

Встроенный Java-код (scriptlet)

Вы можете встраивать действительные Java-операторы стандартного языка Java непосредственно внутрь JSP-файлов. Этот код должен заключаться между тегами <% и %> . Такой встроенный код называется *скриптлетом* (scriptlet).

Скриптлет для сервлетов может использовать набор предопределенных переменных, которые соответствуют определенному сервлету, вводному и выводному классам:

- **request** — класс запроса для сервлета, определен в javax.servlet.http.HttpServletRequest;
- **responses** — класс ответа для сервлета, определен в javax.servlet.http.HttpServletResponse;
- **out** — выводной класс, определен в java.io.PrintWriter;
- **in** — вводной класс, определен в java.io.BufferedReader.

Пример:

```
<%
foo = request.getParameter("Name");
out.println(foo);
%>
```

Совместное использование JSP и XML

Главным отличием JSP-файлов от обычных HTML-файлов является то, что они могут исполняться динамически — например, они могут обрабатывать формы или читать базы данных — используя Java в качестве скриптового языка на стороне сервера. С какой стороны тут подверстать XML, на первый взгляд, понять трудно. Однако можно использовать XML для хранения данных на Web-сервере, а JSP для отображения этих данных.

Вообще, появление XML — это серьезный выпад в сторону реляционных баз данных в эпоху глобальных информационных ресурсов и Интернета. А, в самом деле, чем XML хуже? Кто может поручиться, что в будущем подавляющее количество информации, в том числе и нормализованной, будет храниться не в реляционных базах данных, а в формате XML. К этому есть серьезные предпосылки.

- XML — это обычный текстовый формат, который легко распечатать и хранить на бумаге, а не на электронном носителе. Что будет с современными электронными носителями через 100 лет — одному богу известно. Попробуйте найдите компьютер, который способен прочитать магнитную ленту в ЕС-формате, на которых еще двадцать лет тому назад хранилось подавляющее количество информации. Да что там магнитная лента, скоро вы не найдете даже устройства, считывающего информацию с флоппи-дисков формата 1,2 Мбайт. Поскольку текстовый формат поддается даже визуальной расшифровке, всегда можно поручиться, что его могут прочитать и понять любые устройства на сколько бы поколений вперед они от нас не отстояли.
- Файл XML легко редактировать обычным текстовым редактором, не прибегая к запуску каких-то там сложных утилит из арсенала баз данных.
- Не надо поддерживать отдельные процессы на сервере для взаимодействия с базами данных, не нужно изучать SQL и писать SQL-запросы, которые конвертируют данные из реляционной в объектную структуру и обратно. Если вы храните XML-файлы, вам нет необходимости использовать подобную надстройку.
- Преимуществом XML является то, что будучи иерархичнее реляционного формата, XML может быть использован для конструирования структуры данных значительно более явным образом. Вам не нужно применять формулы отношений элементов или нормализовывать вашу модель. Если

у вас один элемент содержит другой, вы можете отразить это непосредственно в файле, а не использовать совместную таблицу.

Разумеется, базы данных останутся, так как в вычислительном мире без них просто невозможно обойтись, потому что некоторые виды работ именно требуют изощренной нормализации, транзакционности, многопотоковости, параллельности, управления доступом, мощнейших языковых средств для формирования запросов к данным, типа PL/SQL и т. д., и т. п. Однако существует масса более простых повседневных задач управления информационным контентом, где применение суперсерверов СУБД просто излишне. Для таких случаев и следует воспользоваться возможностями JSP и XML.

Предположим я завожу у себя на компьютере свой собственный домашний музей художественных произведений, образы которых во множестве доступны из Интернета. Музей отражает, естественно, только лишь ваши предпочтения, как коллекционера, и может хранить и образы, и сопроводительные комментарии к ним в одном каталоге, оформленные в виде соответствующего XML-описания, например:

```
<picture>
    <title>Изумительное, чисто американское искусство Хоппера</title>
    <date>Период расцвета 1930-1940</date>

    <caption>Городской цикл</caption>
    <image>
        <src>hopper.hotel-room.jpg</src>
        <width>340</width>
        <height>200</height>
    </image>
    <thumbnails>
        <image>
            <src>hopper.morning-sun.jpg</src>
            <width>72</width>
            <height>72</height>
        </image>
        <image>
            <src>hopper.women-sun.jpg</src>
            <width>120</width>
            <height>80</height>
        </image>
    </thumbnails>
</picture>
```

Теперь возникает вопрос: каким образом можно использовать XML-данные в JSP-странице? Существует несколько способов для этого.

- **DOM.** Вы можете использовать классы, реализующие DOM-интерфейс, чтобы просматривать и проверять XML-файл.

- **Xpath.** Вы можете использовать Xpath-процессор, чтобы определить местоположение элементов в XML-файле.
- **XSL.** Вы можете использовать XSL-процессор, чтобы трансформировать XML в HTML.
- **Собственный Bean-компонент.** Вы сами можете написать внешний класс.

DOM (Document Object Model)

В данном разделе мы только кратко коснемся методов разбора и интерпретации содержимого XML-файлов, потому что это отдельная и большая тема, о которой более подробно будет говориться в следующей главе. Существуют специальные парсеры, которые могут трактовать содержимое XML-файлов с использованием объектно-ориентированной модели на основе Java-классов. DOM является аббревиатурой *Document Object Model.* Это стандартный API для просмотра документов XML, разработанный World Wide Web Consortium (W3C). Существует много реализаций DOM-парсера. Один из них — XML4J от IBM, но вы можете использовать любой DOM-парсер, например, фирмы Sun. Это возможно потому, что DOM является набором интерфейсов, а не классов — и все DOM-парсеры возвращают объекты, которые должным образом реализуют эти интерфейсы.

Простой XML-файл для коллекционных изображений, описанный выше, представляется в DOM несколькими объектами в древовидной структуре.

```
Document Node
  --> Element Node "picture"
            --> Text Node "\n   " (пробел)
            --> Element Node "title"
                --> Text Node "Изумительное, чисто американское..."
            --> Element Node "date"
           --> ... и т. д.
```

Чтобы получить значение "Изумительное, чисто американское...", вам необходимо выполнить несколько вызовов методов из объекной модели DOM. К сожалению, эта модель не лишена недостатков, она слишком запутана и в ней много лишнего. В то же время в DOM, как ни странно, не существует метода, с помощью которого можно было бы найти конкретное текстовое значение элемента с конкретным названием. XML-объекты, видимые через DOM, непосредственно недоступны, доступ к ним обеспечивается только порциями, посредством DOM API. В последнее время этот недостаток DOM активно преодолевается программистами. В качестве примера можно сослаться на статью Алекса Чаффи (Alex Chaffee), опубликованную в JavaWorld, из которой я взял основные идеи для данного раздела. Алекс написал свой собственный небольшой вспомогательный класс `DOMUtils`, который содержит static-методы для выполнения типовых задач DOM. Например, чтобы

получить текстовое содержимое подэлемента `title` корневого элемента `picture`, вы должны написать следующий код:

```
Document doc = DOMUtils.xml4jParse(picturefile);
Element nodeRoot = doc.getDocumentElement();
Node nodeTitle = DOMUtils.getChild(nodeRoot, "title");
String title = (nodeTitle == null) ? null :
   DOMUtils.getTextValue(nodeTitle);
```

Получение значения подэлементов изображений также незамысловато:

```
Node nodeImage = DOMUtils.getChild(nodeRoot, "image");
Node nodeSrc = DOMUtils.getChild(nodeImage, "src");
String src =  DOMUtils.getTextValue(nodeSrc);
```

И так далее, т. к. каждому значимому элементу соответствует Java-переменная, все, что вам нужно — это поместить данные переменные внутрь вашей HTML-разметки, используя стандартные теги JSP.

```
<table bgcolor="#FFFFFF" border="0"
       cellspacing="0" cellpadding="5">
<tr>
<td align="center" valign="center">
<img src="<%=src%>" width="<%=width%>"
            height="<%=height%>" border="0"
            alt="<%=src%>"></td>
</tr>
</table>
```

В полном тексте .JSP файла это может выглядеть следующим образом:

Распечатка файла Picture-dom.jsp

```
<%
// Вызов этого файла может быть таким: picture-dom.jsp?file=museum.xml
%>

<%@include file="init.jsp"%>
<%
String picturefile =
application.getRealPath("/" + request.getParameter("file"));

System.out.println(picturefile);

Document doc = DOMUtils.xml4jParse(picturefile);
Element nodeRoot = doc.getDocumentElement();

Node nodeTitle = DOMUtils.getChild(nodeRoot, "title");
String title = (nodeTitle == null) ? null :
DOMUtils.getTextValue(nodeTitle);
```

```
Node nodeCaption = DOMUtils.getChild(nodeRoot, "caption");
String caption = (nodeCaption == null) ? null :
DOMUtils.getTextValue(nodeCaption);

Node nodeDate = DOMUtils.getChild(nodeRoot, "date");
String date = (nodeDate == null) ? null :
DOMUtils.getTextValue(nodeDate);

Node nodeImage = DOMUtils.getChild(nodeRoot, "image");
Node nodeSrc = DOMUtils.getChild(nodeImage, "src");
String src =  DOMUtils.getTextValue(nodeSrc);
Node nodeWidth = DOMUtils.getChild(nodeImage, "width");
String width =  DOMUtils.getTextValue(nodeWidth);
Node nodeHeight = DOMUtils.getChild(nodeImage, "height");
String height =  DOMUtils.getTextValue(nodeHeight);
%>

<html>
<head>
<title><%=title%></title>
</head>

<body>
<center>
<table bgcolor="#CCCCCC" border="0" cellspacing="0" cellpadding="8">
<tr>
<td align="left" bgcolor="#6666AA" width="100%">

<b><%=title%></b>

</td></tr>

<tr>
<td align="center" valign="top">
 <table bgcolor="#FFFFFF" border="0" cellspacing="0" cellpadding="5">
 <tr>
 <td align="center" valign="center">
 <a href="picture-sm-dom.jsp"><img src="<%=src%>" width="<%=width%>"
height="<%=height%>" border="0" alt="<%=src%>"></a></td>
 </tr>
 </table>
</td>
</tr>

<tr>
<td align="center" valign="center">
<% if (caption != null) { %>
  <b><%=caption%></b><br/>
<% } %>
```

```
<% if (date != null) { %>
  <font size="-1"><i><%=date%></i></font><br/>
<% } %>
</td>
</tr>
</table>

</td>
</tr>
</table>


</body>
</html>
```

Полный текст статьи Алекса Чаффи помещен в мартовском номере журнала JavaWorld за 2000 г.

Преимущества данного подхода, как это можно видеть из текстов программ, заключается в том, что в HTML-тегах, включенных в JSP-файл и интерпретируемых любым браузером, можно вставлять объектные переменные, уже получившие значения. При этом преодолевается главный недостаток чистого HTML-формата — это его вавилонское смешение всего на свете: текстов, образов, стилей, программ обработки и т. д. В этом случае, данные выделены отдельно и помещены в XML-структуру, которой весьма легко управлять, ровным счетом ничего не меняя в самом HTML-файле. Несомненно, что в таком подходе просвечивает наше уже весьма недалекое будущее.

Интернационализация и проблемы с кириллицей по отношению к JSP

Хотя и утверждается, что Web-сайты, основаные на JavaServer Pages (JSPs) легче поддаются интернационализации, чем созданные с помощью сервлетов (servlets), на самом деле это далеко не так, особенно если вы работаете с оригинальными JSP-интерпретаторами. Главный посыл этого утверждения заключается в том, что технология JSP способствует отделению дизайна от логики приложения. Однако, смиря свое сердце, следует помнить, что создатели современных программных технологий почти никогда не задумываются над тем, что существуют другие языки, помимо английского, тем более, русский, символы которого сами русскими могут кодироваться 5-ю различными кодировками! Это ли не глупость с точки зрения создателей универсальных технологий, хотя, если задуматься, — на то они и универсальные, чтобы не зависеть ни от каких обстоятельств, тем более от языковых настроек. Изначально в JSP предполагалось, что все сообщения для пользователя, надписи на кнопках, даты, названия денежных единиц и т. д., должны быть выделены во внешние файлы. Такие вхождения, содержащие

разную информацию для разных локальных областей (locale), называют *ресурсами* (resources), а ресурсы, сгруппированные во внешнем файле класса или ресурса, — *ссылками на ресурсы* (resource bundle) или просто *связками* (bundle). Несмотря на то, что вы можете сохранить объекты любого типа как значения, используя класс-файлы, большинство Web-приложений применяют файлы свойств, имеющих расширение .properties. В статье Говинда Сишарди (Govind Seshardi) приводится пример класс-файла, служащего для того же, что и файл свойств:

```
public class Message_ru_RU extends ListResourceBundle {
 public Object[][] getContents() {
  return contents;
}
static final Object[][] contents = {
  {"main.title", "Музыка без границ"},
  {"main.subhead", "Песни мира"},
  {"main.addLabel","Добавить"},
  . . .
  {"cd.quantityLabel","Количество"}
 }
}
```

Благодаря файлам свойств, можно изолировать элементы интерфейса и локализовать их в файлах свойств, а затем использовать в страницах JSP. Покажем сначала 2 примерных файла.

Листинг 1. Message.properties

```
main.title=Music Without Borders
main.subhead=Sounds from the Global Village
main.addLabel=Add
main.qtyLabel=Quantity
main.bgcolor=#33CCFF
cart.bgcolor=#FFFFFF
cart.delLabel=Delete
cart.checkoutLabel=Checkout
checkout.bgcolor=#33CCFF
checkout.title=Music Without Borders Checkout
checkout.subhead=Thanks for your order!
checkout.totalLabel=Total
checkout.returnLabel=Shop some more!
dollar.exchRate=1.00
cd.albumLabel=Album
cd.artistLabel=Artist
cd.countryLabel=Country
cd.priceLabel=Price
cd.quantityLabel=Quantity
```

Листинг 2. Message_ru_RU.properties

```
main.title=Музыка без границ
main.subhead=Песни мира
main.addLabel=Добавить
main.qtyLabel=Количество
main.bgcolor=#33CCFF
cart.bgcolor=#FFFFFF
cart.delLabel=Удалить
cart.checkoutLabel=Заказать
checkout.bgcolor=#33CCFF
checkout.title= Проверка и подтверждение заказа
checkout.subhead=Спасибо за сделанный выбор
checkout.totalLabel=Всего
checkout.returnLabel=Нечто еще!
dollar.exchRate=6.48
cd.albumLabel=Альбом
cd.artistLabel=Артист
cd.countryLabel=Страна
cd.priceLabel=Цена
cd.quantityLabel=Количество
```

Известно, что в Java имеется специальный класс, предназначенный для интернациональной поддержки, — java.util.Locale, который использует код страны и код языка в форме:

```
Locale locale=null;
if (lang.equals("German")) {
locale=Locale.GERMANY;
} else if (lang.equals("Russian")) {
locale=Locale.RUSSIAN;
} else if (lang.equals("Russian")) {
locale=new Locale("ru","RU");
} else {
locale=Locale.US;
}
```

Код страны — это две буквы, набранных в верхнем регистре, а код языка — две буквы в нижнем регистре.

Как только локальная область (`locale`) создана, вы можете связать ее с соответствующим классом (см. аналог файла свойств — пример класса Говинда Сишарди):

```
ResourceBundle bundle =
ResourceBundle.getBundle("Message", locale);
```

Вы можете хранить эти классы где угодно внутри `CLASSPATH`, который использует сервер приложений, поддерживающий технологию JSP. Вызов

getBundle() начинает поиск связки после выделения кода языка (ru) и кода страны (RU) из объекта locale, переданного как параметр. Сначала ищется класс Message_ru_RU.class, но если он отсутствует, ищется класс Message_ru.class и лишь потом уже Message.class. В случае неудачной попытки, осуществляется поиск соответствующего файла свойств, начиная с Message_ru_RU.properties, переходя к Message_ru.properties и заканчивая Message.properties. Если и эта попытка неудачна, функция getBundle() выдает в качестве результата MissingResourceException.

Существует и еще одна проблема при работе с кириллицей. Дело в том, что для всех западноевропейских языков обычно используется стандартный 8-битовый набор кодируемых символов ISO Latin-1 (ISO-8859-1). Хотя он и позволяет кодировать 256 различных знаков, символов кириллицы в нем, к глубокому сожалению, не имеется, и нам приходится применять другой стандарт ISO-8859-5, а для этого — маскировать различия между ISO-8859-5 и ISO-8859-1, указывая ISO-8859-5 явно, потому что во многих случаях ISO-8859-1 принят по умолчанию. Например, при использовании JSP для генерации динамического содержания Web-страниц, вы должны сообщить вашему браузеру тип кодировки в форме утверждения:

```
<%@ page contentType="text/html; charset=ISO-8859-5" %>
```

Неудобство заключается в том, что не все версии браузеров поддерживают этот тип кодировки. Вы должны стремиться использовать только те браузеры, которые полностью понимают стандарт HTML 4.0, потому что именно в этом стандарте заложена возможность поддержки большинства всемирных азбук и языков.

Методы загрузки Java-кодов, сервлетов и апплетов

Нововведения в концепции апплетов и загружаемого по сети ПО со стороны Microsoft

Инновации Microsoft

Microsoft разработала концепцию по загрузке из Интернета программ и способа их автоматической инсталляции на компьютере пользователя. Ключевыми терминами этой концепции являются следующие:

Компоненты

ActiveX® Control (.ocx), .dll, .exe, Java классы и апплеты.

Пакетирование (packaging)

Для более быстрой загрузки по сети и связывания используемых компонентов их рекомендуется пакетировать. Пакетирующим механизмом обычно

служит архиватор, который порождает архивный файл (a cabinet file), адресуемый при помощи CODEBASE-атрибута в теге OBJECT на Web-странице.

Кабинетные файлы (cabinet files)

Этот архивный формат позволяет сохранить множество файлов в одном файле с расширением (.cab). Данный формат понимают браузеры и самостоятельно распаковывают его.

Microsoft вместе с ОС поставляет специальные средства для создания и чтения таких файлов. Кабинетные файлы могут быть и, как правило, должны быть подписаны.

Цифровая подпись (digital signature)

Загрузка по Интернету и размещение на своем компьютере чужой программы всегда сопряжены с большой опасностью, особенно если это не Java-апплет (почти всегда безопасный, потому что он работает только в контексте JVM и не может выпрыгнуть за ее пределы), а ActiveX-компонент, которому могут быть доступны все ресурсы операционной системы. Пройдя за два десятилетия своего существования через множество неприятностей, связанных с безопасностью, Microsoft предлагает пользователям снабжать цифровой подписью все разрабатываемые ими программные элементы, разработав для этого целую группу специальных программных средств.

Цифровая подпись (digital signature) связывает имя разработчика с данным файлом и обеспечивает специальную учетную запись, в которой можно задать пароль. Когда уровень максимальной защиты установлен в браузере, как уровень по умолчанию, то все загружаемые по сети элементы (т. е. присутствующие в теге OBJECT) должны быть подписаны.

С точки зрения Microsoft, апплетом называется любая программа, которую можно загрузить и выполнить в среде браузера, такого, например, как MS Internet Explorer. Учитывая массу критических отзывов о недостаточно надежной системе защиты при применении Интернет/интранет-технологий в исполнении Microsoft, фирма разработала стройную концепцию аутентификации программ с помощью цифровой подписи. Подписать можно исполняемые программы следующих типов (указаны расширения файлов):

- .cab
- .dll
- .cat
- .exe
- .ctl
- .ocx

Программы для аутентификации, которые разработала Microsoft, следующие:

- MakeCert — создает X.509 сертификат;
- Cert2SPC — создает Software Publisher Certificate (SPC);
- SignCode — подписывает файл;

- ChkTrust — проверяет подлинность подписи в файле;
- MakeCTL — создает список цифровых сертификатов;
- CertMgr — управляет сертификатами и списками сертификатов;
- SetReg — устанавливает специальные регистровые ключи, необходимые для процесса сертификации;
- MakeCat — создает объединенный каталог файлов, чтобы избежать повторных проверок и повторного запуска механизма аутентификации в MS Internet Explorer.

Все эти программы можно свободно скачать с фирменного сайта Microsoft или получить через MSDN.

INF-файлы

Информационные файлы (.inf) содержат инструкции по инсталляции, которые используются в Internet Explorer для корректной инсталляции и регистрации загружаемой программы.

INF-файл содержит в себе неограниченное количество именованных секций, каждая из которых имеет определенное содержание. Например, можно определить секции для осуществления функции копирования файлов на диск и секции для добавления соответствующих записей в системный реестр.

Файлы OSD (Open Software Description)

Open Software Description, а попросту говоря, описание ПО — это XML-структура, которая описывает программные компоненты и их связи между собой. OSD — это последнее по времени изобретение Microsoft, в котором она практически применила новомодную технологию XML, о которой подробно описано в этой книге.

OSD обеспечивает широкий и более гибкий словарь, чем INF-файлы для описания связанных сетью данных и распределенного программного обеспечения, — в этом и заключается инновационный подход Microsoft. Когда структура и взаимозависимости модулей могут быть ясно описаны на XML, в будущем при повторной загрузке пакета будут загружаться только те его части, которые должны быть изменены. Это чрезвычайно экономный способ постоянного обновления программного обеспечения по Интернету, что-то вроде репликация в Lotus Notes.

Java-классы и апплеты

В отличие от INF-файлов, OSD-файлы поддерживают вложение Java-классов и апплетов в дистрибутивные пакеты программ, загружаемых по Интернету. Для этого в части OSD, описывающей XML-структуру, предусмотрены специальные теги JAVA и PACKAGE.

Языковая поддержка

OSD-словарь обеспечивает способ, который гарантирует, что при загрузке ПО будут соблюдаться правила его развертывания на машине пользователя в родных ему кодировках. Для этого также предусмотрены специальные конструкции в XML-описании, такие как DEPENDENCY, IMPLEMENTATION или элементы SOFTPKG.

Операционная система и аппаратные платформы

Другие важные элементы языка, такие как OS и PROCESSOR, также уменьшают риск некорректной инсталляции, если пользователь при загрузке той или иной версии ПО что-нибудь перепутал.

Примеры использования

OSD-файлы используются, чтобы описать части структурной иерархии пакетов программ и отобразить зависимость частей друг от друга. OSD-файл может быть сжат и представлен в виде файла .CAB, дистрибутивного архива ПО. OSD-словарь может использоваться, чтобы описать Java-классы и апплеты, а также сложные зависимости в структуре программного обеспечения, требования языка и платформы.

Microsoft придумала еще так называемый *формат определения канала* (Channel Definition Format — CDF).

Каналы модификации программного обеспечения

Формат определения канала (CDF) — это XML-словарь для отображения и заполнения определенных страниц на Web. Они используются, чтобы описать иерархию страниц на Web-сайте, а также для постоянной модификации их содержимого. Web-сайты, описанные CDF-файлами, упоминаются как активные каналы Web.

Пользователи "подписываются" на активные каналы, просто нажимая гиперссылки на страницах Сети, по которой загружаются CDF-файлы. Подписка дает шанс пользователям просмотреть содержание канала автономно, если они не считают необходимым соединение по модему. У них всегда остается возможность получать регулярные автоматические модификации содержания канала. Активные каналы способны извещать подписчиков относительно новых версий программного обеспечения, которое пользователь уже установил у себя. Все это также базируется на идеологии XML.

Естественно, что в связи с введением этой технологии претерпел некоторые изменения и тег OBJECT. Теперь в HTML-файле он может выглядеть следующим образом:

```
<OBJECT
    ACCESSKEY=key
```

```
    ALIGN=ABSBOTTOM | ABSMIDDLE | BASELINE | BOTTOM | LEFT |
        MIDDLE | RIGHT | TEXTTOP | TOP
    CLASS=classname
    CLASSID=id
    CODE=filename
    CODEBASE=url[#version=a,b,c,d]
    CODETYPE=media-type
    DATA=url
    DATAFLD=colname
    DATASRC=#ID
    HEIGHT=n
    ID=value
    LANG=language
    LANGUAGE=JAVASCRIPT | JSCRIPT | VBSCRIPT | VBS
    NAME=name
    STYLE=css1-properties
    TABINDEX=n
    TITLE=text
    TYPE=MIME-type
    WIDTH=n
    event = script
>
```

Примечание

Тег OBJECT может появляться в тегах HEAD или BODY HTML-документа.

Примечание

События могут быть посланы непосредственно объекту OBJECT или обрабатываться специальным обработчиком в скрипте.

Примечание

Объект OBJECT — это блок элементов и должен быть оформлен как закрытый тег.

Со стороны Microsoft подвергся ревизии также и элемент CODEBASE.

Он имеет следующий синтаксис:

```
<CODEBASE
    FILENAME="file"
    HREF="url"
    SIZE="n"
    STYLE="ActiveSetup" | "MSICD"
/>
```

Атрибуты

`FILENAME` — строковое значение имени дистрибутивного файла (.cab file) как OSD-файл.

`HREF` — URL файла или дистрибутивного пакета файлов. Когда используется атрибут `HREF`, предполагается, что файл загружается по сети.

`SIZE` — число в килобайтах, которое определяет максимальный размер дистрибутива. Если размер превышен, дистрибутив не будет загружен. Если размер не определен, никакое ограничение не предписывается. Этот атрибут допустим только внутри CDF-файлов.

`STYLE` — строка, определяющая механизм для загрузки и инсталляции файлов по сети Интернет. Этот атрибут допустим только внутри CDF-файлов. Может быть два значения.

`"ActiveSetup"` — используется "Active Setup"-машина (engine), т. е. старая технология.

`"MSICD"` — применяется технология Microsoft Internet Component Download (MSICD) для загрузки и инсталляции ПО в соответствии с принципами Open Software Description (OSD). Для использования с Microsoft Internet Explorer 3.0 должен быть обеспечен inf-файл.

Безопасность в Интернете — подписанные апплеты

Методы фирмы Sun, предлагаемые для распространения по сети загружаемого и безопасного программного обеспечения

Изначально, когда только создавался язык Java, апплеты, в целях безопасности, которой всегда уделяла особое внимание фирма Sun, были весьма ограничены в своих возможностях по сравнению, например, с OCX-модулями, которые тогда активно пропагандировала Microsoft (известно, к каким грандиозным сетевым скандалам, ударив по репутации самой Microsoft, это впоследствии привело!). Так вот, обычный апплет по сравнению с обычным приложением ограничен во многом:

- апплет не может читать, писать и выполнять файлы на локальной машине;
- апплет не может открыть сетевое соединение с сервером, исключая тот, с которого он был загружен;

- апплету доступна ограниченная информация о локальной машине (например, символ-разделитель в именах файлов, версия виртуальной Java-машины и т. д.);
- загруженный апплет не имеет доступа к существующей на локальной машине иерархии Java-классов, а может использовать только "встроенную" в браузер версию.

Таковы были требования документа NS 2.0/IE 3.0 и, если кто не верит, может внимательно изучить его. Впоследствии, естественно, появилась идея "подписанных" (signed) апплетов, для которых эти ограничения снимаются. Все подробности с примерами и описаниями изложены в основополагающих документах разработчиков Java на сайте:

http://developer.java.sun.com/developer/technicalArticles/Security/Signed /index.html

В этой книге мы только кратко коснемся данных проблем, потому что помимо апплетов существует огромное количество решений, в том числе и на Java, чтобы обойти эти ограничения при проектировании распределенных приложений в среде интранета.

Процедура создания "подписанных" апплетов состоит в следующем:

- все class-файлы апплета упаковываются в Java-архив (файл с расширениеем .jar). В архив можно упаковать и все файлы .gif и .jpg, используемые в апплете что существенно уменьшает объем данных, которые передаются при загрузке апплета;
- разработчик создает пару ключей для апплета — личный (private, закрытый) и публичный (public, открытый);
- разработчик "подписывает" апплет, добавляя в файл .jar специальную информацию, содержащую версию личного ключа;
- разработчик создает файл-сертификат, который содержит информацию об открытом ключе, но может быть передан по сети без дополнительного шифрования;
- разработчик передает пользователю файл-сертификат. Обычно это делается через ftp или e-mail;
- пользователь выполняет определенную (довольно простую) процедуру установки файла-сертификата на своей машине. Этот процесс называется create the policy file.

Как только браузер получает по сети подписанный апплет, он немедленно может удостовериться, что данному апплету можно доверять. Если процедура подтверждения заканчивается успешно, апплету предоставляются в распоряжение список его прав (permissions) на машине. Задание прав можно осуществить специальной утилитой policytool в JDK1.2.

Кратко шаги по созданию подписанного апплета можно определить как следующие (используются утилиты из состава JDK1.2).

Компиляция апплета

```
javac SignedAppletDemo.java
```

Создание архивного файла

```
jar cvf SignedApplet.jar SignedAppletDemo.class
```

Создание ключей (keys)

Создадим базу данных с именем, например, susanstore, как это описано в основополагающем руководстве на **developer.java.sun.com**. В этой базе при помощи утилиты keytool будет сгенерирована пара ключей (public и private) с общим ключом в сертификате. JAR-файл подписывается личным (private) ключом и может быть проверен получателем через предоставляемый ему файл сертификата с общим ключом, чтобы проверить подлинность подписи, т. е. личного ключа владельца. Вот как рекомендуется использовать утилиту keytool.

```
keytool -genkey -alias signFiles -keystore susanstore -keypass
kpil35 -dname "cn=jones" -storepass ab987c
```

`-keypass kpil35` — это пароль, чтобы обратиться к сгенерированному секретному (private) ключу в паре. Пароль для доступа к базе данных susanstore, которая создается в текущем каталоге также определяется по `-storepass ab987c`.

Опция `-dname "cn=jones"` специфицирует различаемое имя в стандарте X.500 для идентификации входа при помощи сертификата стандарта X.509.

Вы можете посмотреть все опции этой утилиты по команде:

```
keytool -help
```

Подпись JAR-файла

Для этого надо использовать утилиту jarsigner из состава JDK:

```
jarsigner -keystore susanstore -storepass ab987c -keypass
kpil35 -signedjar SSignedApplet.jar SignedApplet.jar signFiles
```

Заметим, что пароль `kpil35` — это не сам секретный ключ, а только пароль для доступа к нему.

Экспорт сертификата с общим ключом

Сертификат с публичным ключом должен быть послан тому, кто собирается использовать этот апплет. Он нужен только для того, чтобы убедиться, что подписанный JAR-файл тот самый, ибо лишь в нем и в присланном сертификате имеется совпадающая пара секретного ключа. Извлечь этот серти-

фикат для посылки его к пользователю из базы данных susanstore можно по команде:

```
keytool -export -keystore susanstore -storepass ab987c -alias
signFiles -file SusanJones.cer
```

Вот теперь можно отправлять файл SusanJones.cer к Джону.

Импорт сертификата

Джон получает по почте или каким-нибудь иным способом сертификат и JAR-файл с подписанным апплетом. Он импортирует его в свой домашний каталог также по команде `keytool`.

```
keytool -import -alias susan -file SusanJones.cer -keystore
jonestore -storepass abcdefgh
```

Создание policy-файла

Этот файл содержит список прав, который можно предоставить апплету, после окончательного удостоверения его подлинности. Например, в нижеследующей процедуре определяется, что апплету позволено создать новый файл в домашнем каталоге пользователя. Policy-файл можно создать либо с помощью утилиты policytool, либо с помощью обычного текстового редактора.

```
keystore "/home/ray/raystore";

// Простейший policy-файл позволяет JavaTM-программе
// создать новый файл в домашнем каталоге пользователя
// Serg Dunaev

grant SignedBy "susan" {
        permission java.util.PropertyPermission
                "user.home", "read"
        permission java.io.FilePermission
                 "${user.home}/newfile", "write"
};
```

Сохраним этот файл под именем my_policy.s.

Выполнение апплета

Подготовленный апплет можно выполнить из браузера, поддерживающего цифровые сертификаты (в MS Internet Explorer для этих целей существует средство Secutity Manager), или при помощи команды `appletviewer`, на входе которой указывается имя с заготовленным списком прав, предоставляемых апплету в policy-файле.

```
appletviewer -J-Djava.security.policy=my_policy.s
http://your.domain.com/SignedApplet.html
```

Приложение к главе 4. Интернет-ресурсы и источники

Протоколы Интернета

ftp://ftp.ripe.net

http://www.ds.internic.net/rfc/rfc-index.txt

http://www.kiarchive.ru:/pub/internet/

http://www.csl.sony.co.jp/rfc/mirror/

http://www.rfc.fh-koeln.de/doc/rfc/html/

Полное описание протокола HTTP 1.1:

http://info.internet.isi.edu/in-notes/rfc/files/ rfc2068.txt.

Безопасность

Документацию по безопасной работе JDK 1.2 можно найти на **http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html**. Здесь находятся подробные сведения по защите информации, включая архитектуру, учебники, API и полный комплект документации.

Информация об использовании криптографии Java Cryptography Extension (JCE) см. **http://java.sun.com/products/jce/index.html**.

Информацию по Java Plug-in, загружаемой в качестве новейшей версии Java Runtime Environment-совместимой с JDK 1.2, можно найти на **http://java.sun.com/products/plugin/index.html**.

Скриптлеты

Creating XML Recordsets — описываются технологии применения языка XML при программировании Web-приложений с использованием технологий Microsoft:

http://www.asptoday.com/articles/19990318.htm

Как работают XML скриптлеты? (XML Scriptlets 1/2 : How they Work):

http://www.asptoday.com/articles/19990426.htm

Использование DHTML-скриптлетов (To use DHTML Scriptlets or not to use DHTML Scriptlets...?):

http://www.asptoday.com/articles/19991027.htm

Сервлеты

Журналы "Открытые системы" и "СУБД":

http://www.osp.ru

Сервер разработчиков Java:

http://java.sun.com

http://developer.java.com

Доступ к БД из сервлета (Владислав Каменский):

http://muma.tusur.ru/docs/java/javapower/faq/servlet/faq.htm

Создание Java-сервлетов (перевод: Андрей Чернухо):

http://java.iba.com.by/javaweb/ibajavat.nsf/lnarticlesview /418725A12AA1AA62422568FF00375D9B

Динамическая графика в Java-сервлетах (перевод: Виктор Смирнов):

http://www.javable.com/javaworld/05_00/01/

Web технологии. Java servlets. Идеология, настройка, использование:

http://www.chat.ru/~k_v_v/javapower/java/7.htm

Servlet mini-FAQ:

http://muma.tusur.ru/docs/java/javapower/faq/servlet/faq.htm

Проблемы с безопасностью на сервере JavaWebServer для Windows NT фирмы Sun:

http://www.hackzone.ru/msdos/nsp/info/www/secfaq/rus/wwwsf8.html#Q75

XSQL Servlet:

http://www.alee.com/cgi-bin/xslt.pl?article_part=050700/chapters/7

Гномы серверов (Дмитрий Рамодин):

http://athena.vvsu.ru/docs/c-java/servlets/140.htm

Книги и статьи

1. L. Hohmann "Journey of the Software Professional". — Prentice Hall, 1996.
2. M. Cusumano, R. Selby "Microsoft Secrets". — The Free Press/Simon & Schuster, NY, 1995.
3. M. Cusumano, R. Selby "How Microsoft Builds Software". — Communications of the ACM, Vol. 40, N. 6, 1997, pp. 53-61.
4. M. Cusumano, R. Selby "Microsoft's Weaknesses in Software Development". — American Programmer, Vol. 10, N. 10, 1997.
5. S. Knepper, D. Ichbiah "The Making Microsoft". — Prima Publishing, 1993.
6. S. Levy "Hackers: Heroes of the Computer Revolution". — Anchor/Doubleday, NY, 1984.
7. "Software Engineering Standard Collection". — IEEE Press, 1994.
8. IEEE Software, Vol. 10, N. 4, July 1993.
9. Волков Д., Гавердовский А., Косякин И. "Заметки о Российском программировании". — Открытые Системы, № 3, 1997.
10. A. Davis "Fifteen Principles of Software Engineering". — IEEE Software, Vol. 11, N. 6, 1994, pp. 94-101.
11. B. Boehm "A Spiral Model of Software Development and Enhancement". — Computer, Vol. 21, N. 5, 1988, pp. 61-72.
12. G. Booch "Object Solutions". — Addison-Wesley, 1996.
13. E. Yourdon "Who Has the Right Stuff?". — Software Development, N. 9, 1997.
14. "Good-Enough Software". — Application Development Strategies, N. 1, 1996.
15. S. Hamilton "Inside Microsoft Research". — Computer, Vol. 31, N. 1, 1998.
16. Вирт Н. "Долой Жирные Программы". — Открытые Системы, № 6 (20), 1996.
17. XML in IE5 Programmers Reference

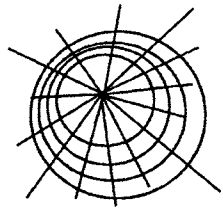
    http://www.amazon.com/exec/obidos/ASIN/1861001576/asptoday
18. Instant DHTML Scriptlets

    http://www.amazon.com/exec/obidos/ASIN/186100138X/asptoday
19. XML Applications

    http://www.amazon.com/exec/obidos/ASIN/1861001525/asptoday

20. XML Design And Implementation
http://www.amazon.com/exec/obidos/ASIN/1861002289/asptoday
21. W3C XML Specification
http://www.w3.org/TR/1998/REC-xml-19980210
22. Microsoft Scriptlets Home Page
http://msdn.microsoft.com/scripting/scriptlets/default.htm
23. Статья "Writing COM Objects with Scripting Languages"
http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/techart/msdn_xmlscripts.htm

ГЛАВА 5



Web-серверы в качестве серверов приложений

Открытые системы и бизнес-решения

Архитектура

Современные бизнес-приложения (e-business applications), предназначенные для развертывания и функционирования в распределенной сетевой среде, так называемые *framework-приложения*, обеспечивают целый набор различных сервисов, основанных на индустриальных стандартах и обладающих способностью к интеграции множества разнообразных компонентов различных фирм-производителей техники и программного обеспечения. На рис. 5.1 показаны ключевые компоненты framework-архитектуры.

Framework-архитектура основана на следующих ключевых элементах:

- тонкие клиенты и апплеты/сервлеты/скриптлеты, позволяющие использовать универсальную модель для доступа к данным;
- основанная на TCP/IP сетевая инфраструктура управления глобальными информационными ресурсами и безопасностью;
- сервер приложений, который обеспечивает платформу для бизнес-приложений и включает сервер HTTP, базу данных, управление транзакциями, почту и средства для обеспечения групповой работы;
- программная среда для разработки Web-приложений, основанная на применении сервлетов Java, EJB и т. д.;
- средства системного управления.

Клиенты

Framework-приложения для бизнес-клиентов основаны, главным образом, на применении тонких клиентов, т. е. таких клиентов, которые используют

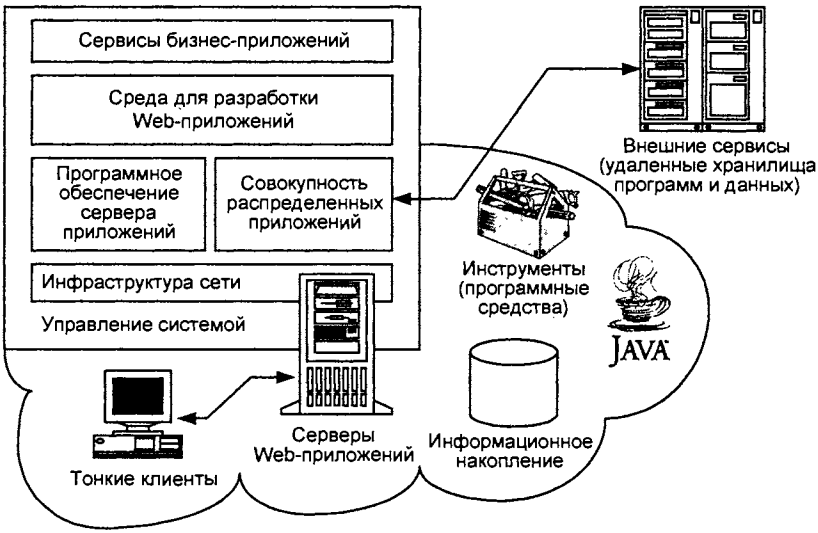


Рис. 5.1. Архитектура framework-приложений (бизнес-приложений)

только стандартное программное обеспечение в виде операционной системы, где ничего не надо инсталлировать дополнительно. В этой модели прикладные программы функционируют на сервере, а по запросу клиентов динамически загружаются и управляются. На самих клиентах могут выполняться распределенные компоненты таких приложений в виде HTML, динамического HTML (DHTML), XML или Java. Подобная чрезвычайно гибкая структура может поддерживать широкий диапазон современных устройств, например, ПК, PDA (карманные компьютеры), смарткарты, сотовые телефоны и пр.

Сетевая инфраструктура

Сетевая инфраструктура для бизнес-приложений должна быть основана исключительно на применении открытых протоколов и стандартов:

- TCP/IP и сетевые сервисы, подобные DHCP, которые автоматически назначают IP-адреса присоединившимся к сети устройствам;
- система шифрования с открытым ключом;
- система управления глобальными директориями, подобная системам NDS, LDAP, Active Directory.

Компонентная интеграция

Компонентная интеграция позволяет разделять приложения, написанные на различных языках программирования и построенные для различных архитектур так, чтобы они прозрачно взаимодействовали между собой.

Компонентная интеграция позволяет Web-клиентам и серверам работать вместе с одними и теми же данными при совершенно различном подходе к проектированию программ, которые никак не должны зависеть от самих данных. Существует несколько методов интеграции.

- *Организация доступа к данным при помощи коннекторов.* Коннекторы прозрачно связывают Web-серверы с различными источниками данных, такими как серверы СУБД, ERP-системы или обычные текстовые файлы. При этом для каждого особого источника данных существует свой собственный коннектор, который реализует специфически-ориентированный протокол. Сами программные средства Web-сервера используют протокол верхнего уровня, единый для связи с любым коннектором, аналогичный протоколу ODBC.
- *Система управления сообщениями*, позволяющая эффективно организовать межзадачный обмен между приложениями, выполняющимися на различных платформах.
- *Объектный сервис*, дающий возможность легко управлять объектно-ориентированными сервисами, используемыми на серверах приложений.

Программные средства для создания Web-приложений

Приведем пример замечательной архитектуры фирмы IBM для разработки и сопровождения Web-приложений (рис. 5.2), основанной на использовании Java-сервлетов, сервисов Enterprise Java и компонентов Enterprise JavaBeans, обеспечивающих функциональную среду для написания безопасных, динамических, транзакционных бизнес-приложений для серверов приложений на базе обычных Web-серверов. Эту архитектуру, как мы впоследствии увидим, можно использовать не только при интеграции программных продуктов IBM, но и при построении Web-приложений на базе открытого кода Linux и Apache. Главное в этой архитектуре — это повышенное внимание к проблемам безопасности, использование многопроцессорных кластерных систем и систем, работающих параллельно, обладающих неограниченными способностями к масштабируемости.

Эти архитектуры уже на деле доказали свою жизнеспособность при обработке чудовищного потока транзакций (например, портал Уимблдонского теннисного турнира во время пиковых моментов финальных матчей был вынужден обслуживать более 1 миллиона запросов в минуту!).

Бизнес-приложения

Бизнес-приложения (e-business applications, где буква e от слова electronic, таким образом дословный и полный перевод звучит так: приложения для

электронной коммерции) — это, как правило, легко выстраиваемые приложения, которые проектируются на очень высоком уровне абстракции и состоят из набора настраиваемых программных компонентов. В результате такого подхода можно быстро порождать бизнес-решения для таких сфер, где традиционно применяются средства вычислительной техники — обработка заказов и платежей, предоставление информационных услуг, работа с каталогами товаров и т. д.

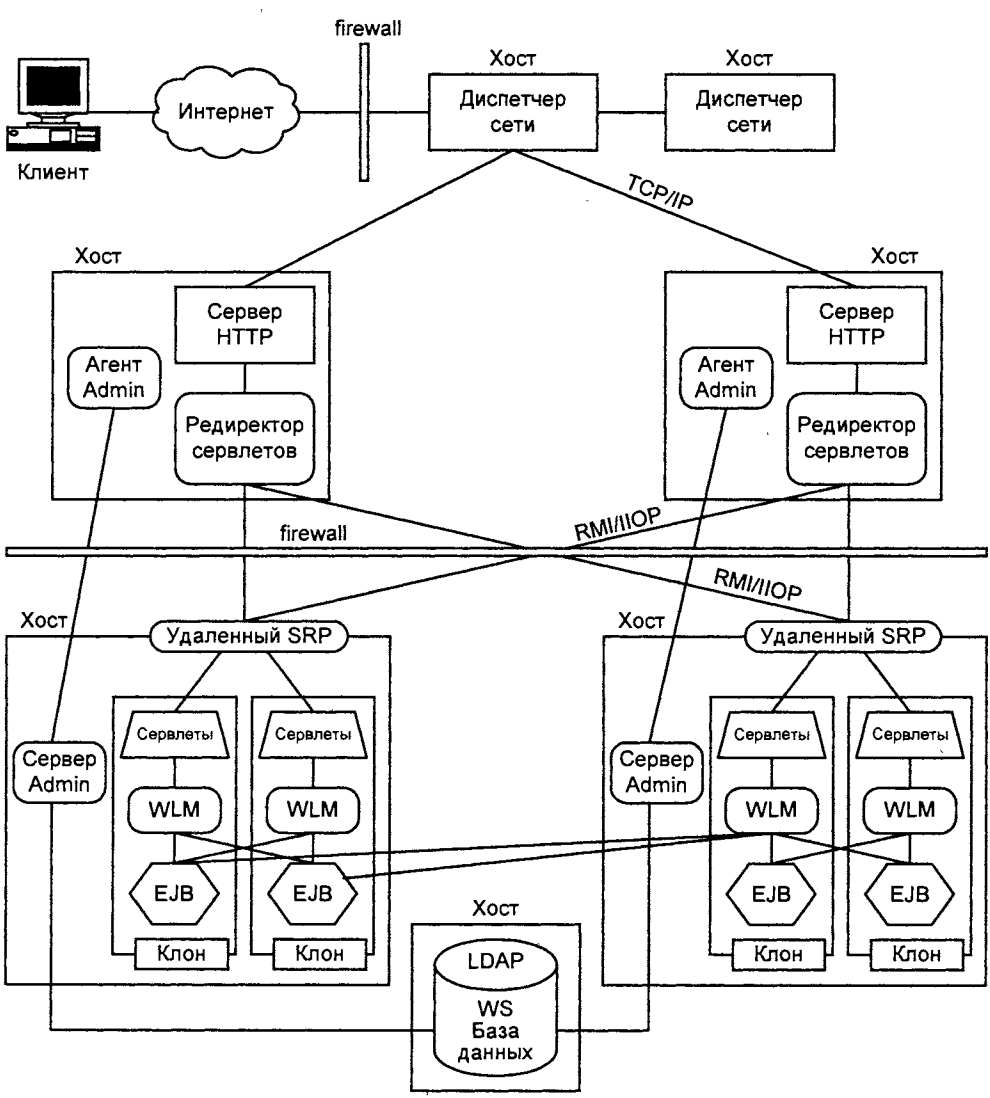


Рис. 5.2. Современные сетевые Web-топологии

Лучшие Web-серверы и серверы приложений

Всемирно известные софтверные компании давно уже заняли на рынке программных средств нишу для серверов приложений, которые почти всегда выполняют роль и обычных Web-серверов. Конкуренция здесь огромная, однако, парадокс нашего времени заключается в том, что эта ниша стремительно расширяется и в бурлящую воронку втягиваются тысячи и десятки тысяч разработчиков без всякой ориентации на сложившиеся стереотипы программирования, потому что Интернет и Интернет-технологии диктуют свои законы. Все фирмы вынуждены придерживаться общепринятых стандартов, поддерживать наиболее популярные технологии, связанные с Интернетом, не говоря уже о гипертекстовых языках, таких как HTML, XML, SGML, отрабатывать удобные средства связи с СУБД и разрабатывать универсальные средства, инвариантные к любым программно-техническим платформам. Лидируют в этой гонке несколько дорогих коммерческих продуктов, среди которых IBM WebSphere, IBM/Lotus Domino Server, Oracle8i и один совершенно открытый и повсеместно доступный некоммерческий проект на базе известного сервера Apache. На этих великолепных разработках, которые гарантируют предотвращение реинжиниринга системы, по крайней мере в течение 10 лет, и содержат в себе наиболее передовые Интернет-технологии, остановимся более подробно.

Серверы-приложений на базе Java и Apache

С точки зрения тех, кто сделал ставку на собственные разработки, начиная с нуля, вплоть до использования операционной системы с открытым кодом, типа Linux, — для таких разработчиков лучше Apache не подойдет ничего, при создании высокоэффективных Web-приложений.

О великолепном Web-сервере Apache, ничуть не уступающим по своим возможностям, а в чем-то даже превосходящим все известные коммерческие Web-серверы, в этой книге уже говорилось немало. Но дело в том, что нынешний Apache легко превратить в мощный многофункциональный сервер приложений, построенный на самых передовых Java-технологиях. Для этого уже существует ряд открытых проектов, доступных в исходных кодах (а как же иначе!), среди которых почетное место занимают Enhydra и Jakarta. Любые группы разработчиков и даже программисты-индивидуалы могут с легкостью присоединиться к этим поистине интернациональным проектам.

Jakarta

В рамках проекта Jakarta уже реализована группа проектов под общим названием Tomcat: в целях использования Java-сервлетов и Java Server Pages (JSP) для создания Web-приложений на базе Apache.

Tomcat хорош тем, что:

- поддерживает наиболее свежие версии спецификации JSP и сервлетов;
- одобрен Sun и Apache;
- может быть запущен без Web-сервера;
- является открытым проектом.

Tomcat

Tomcat — это контейнер сервлетов и JSP-страниц, готовых для выполнения при запуске их из браузера. Он может работать самостоятельно или в соединении с несколькими популярными Web-серверами:

- Apache, версия 1.3 или выше;
- Microsoft Internet Information Server, версия 4.0 или выше;
- Microsoft Personal Web Server, версия 4.0 или выше;
- Netscape Enterprise Server, версия 3.0 или выше.

Иногда спрашивают — в чем разница между Jserv для Apache и Tomcat? Принципиальная разница между ними в том, что Jserv — это совместимый с Servlet API 2.0 контейнер, который был создан для использования в Apache. Tomcat — это полностью переписанный Servlet API 2.2 и совместимый с JSP 1.1 контейнер, который может работать не только с Apache, но и с другими Web-серверами.

Таким образом, в руках разработчиков оказывается мощное программное средство и набор инструментальных средств, с помощью которых можно с максимальной эффективностью реализовать в своих программах и скриптах, предназначенных для использования в Интернет-среде, все функциональные богатства современных информационных технологий, которые отражены в целом ряде передовых спецификаций и стандартов, таких как JSP, XML, EJB и т. д.

Большим преимуществом Tomcat по сравнению с другими продуктами является тот факт, что его можно использовать как встроенное средство в различных средах программирования на Java, например, в такой популярной и, пожалуй, наиболее мощной на сегодняшней день среде, как VisualAge for Java производства фирмы IBM.

Хотя VisualAge for Java Professional и Enterprise Editions содержат в себе средства для программирования JSP 1.0, можно поставить у себя бесплатные Apache и Tomcat и настроить на работу с ними гораздо более дешевую и компактную версию VisualAge for Java Entry Edition, чтобы работать уже с гораздо более развитой версией JSP 1.1 и отладчиком JSP, входящим в структуру Tomcat. Кроме того, и ваши сервлеты будут выполняться с использованием более новой спецификации Servlet 2.2.

Версию VisualAge for Java Entry Edition также можно скачать бесплатно с официального сайта IBM или ее партнеров (**http://www7.software.ibm.com /vad.nsf/Data/Document3793**).

Чтобы скачать Tomcat, воспользуйтесь адресом **http://jakarta.apache.org/**.

Последовательность действий по установке Tomcat из VisualAge for Java

(Раздел основан на материалах статей Шелдона Уосника (Sheldon Wosnick).)

1. Установите VisualAge в каталоге C:\IBMVJava и Tomcat в C:\tomcat.
2. Запустите VisualAge for Java и создайте новый проект с именем Tomcat Version 3.
3. Скопируйте все подкаталоги из C:\tomcat (но не самый главный каталог) в C:\IBMVJava\ide\project_resources\Tomcat Version 3.
4. Нажмите кнопку **Import**.
5. Импортируйте все java и resources в C:\IBMVJava\ide\project_resources \Tomcat Version 3\src.
6. Импортируйте все примеры в C:\IBMVJava\ide\project_resources\Tomcat Version 3\examples.
7. Затем нужно создать два небольших класса для запуска Tomcat их среды Visual Age.

Запуск

```
public class StartTomcat {
  public static void main(java.lang.String[] args) {
    try {
      Class aClass = Class.forName("org.apache.tomcat.shell.Startup");
      java.lang.reflect.Method method = null;
      Class[] argsClass = new Class[] {String[].class};
      // get the main method for the org.apache.tomcat.shell.Startup class
      method = aClass.getMethod("main", argsClass);
      Object object = null;
      Object[] arguments = new Object[] {new String[] {}};
      // now invoke the main method for the
org.apache.tomcat.shell.Startup class
      method.invoke(object, arguments);
    }
    catch (Exception e) {
      System.out.println("Exception occurred: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Останов

```
public class StopTomcat {
  public static void main(java.lang.String[] args) {
    try {
      Class aClass = Class.forName("org.apache.tomcat.shell.Shutdown");
      java.lang.reflect.Method method = null;
      Class[] argsClass = new Class[] {String[].class};
      // get the main method for the org.apache.tomcat.shell.Shutdown class
      method = aClass.getMethod("main", argsClass);
      Object object = null;
      Object[] arguments = new Object[] {new String[] {}};
      // now invoke the main method for the
org.apache.tomcat.shell.Shutdown class
      method.invoke(object, arguments);
    }
    catch (Exception e) {
      System.out.println("Exception occurred:□ " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

1. Создайте подкаталог с именем C:\IBMVJava\ide\project_resources\Tomcat Version 3\classes.

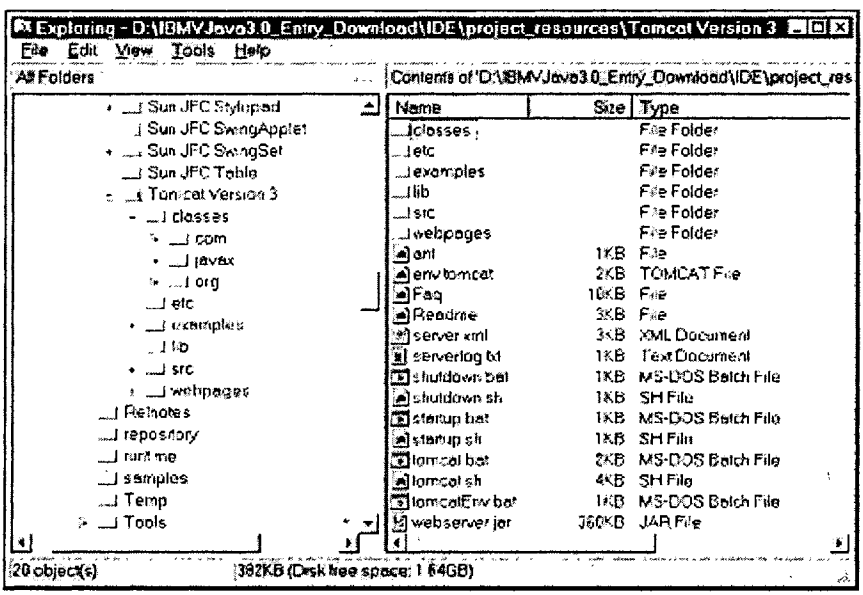


Рис. 5.3. Дерево каталогов IBM Visual Age с Tomcat

2. Разархивируйте в него все классы из следующих архивов Tomcat:
   - webserver.jar;
   - ant.jar;
   - jasper.jar;
   - servlet.jar;
   - xml.jar.

   К этому шагу вы должны иметь загрузочные модули Tomcat и структуру дополнительных классов в Visual Age (рис. 5.3).

Конфигурирование классов Tomcat

1. С помощью правой кнопки выберите класс `StartTomcat`, а затем пункт **Properties** из контекстного меню. В диалоговом окне **Properties** выберите вкладку **Class Path**. Нажмите кнопку **Edit** для редактирования **Extra directories path**.

   Введите (или скопируйте) следующие каталоги к `extra classpath` для класса `StartTomcat`:

   ```
   .\etc\;
   .\examples\WEB-INF\classes\;
   .\webpages\WEB-INF\classes\;
   .\examples\;
   .\webpages\;
   .\work\8080\;
   .\work\8080%2Fexamples\;
   .\classes\;
   ```

2. С помощью правой кнопки выберите класс `StopTomcat`, а затем пункт **Properties** из контекстного меню. В диалоговом окне **Properties** выберите вкладку **Class Path**. Нажмите кнопку **Edit** для **Extra directories path**.

   Введите (или скопируйте) следующие каталоги к `extra classpath` для класса `StartTomcat`:

   ```
   .\classes\;
   ```

   После этих манипуляций Workbench с вашим проектом Workspace Tomcat Version 3 должен содержать нужные пакеты с классами (рис. 5.4).

Фактически интеграция Tomcat и Visual Age for Java завершена.

Выполнение Tomcat

Для того чтобы запустить и остановить Tomcat, имеются две специальные программы: StartTomcat и StopTomcat.

Запустите Tomcat, выбрав программу StartTomcat и нажав кнопку **Run** на панели, или выберите последовательно команды **Selected**, **Run**, **Run Main**. Чтобы погасить или остановить Tomcat, надо запустить программу StopTomcat с помощью кнопки **Run** или выбрать последовательно команды **Selected**, **Run**, **Run Main** (рис. 5.5).

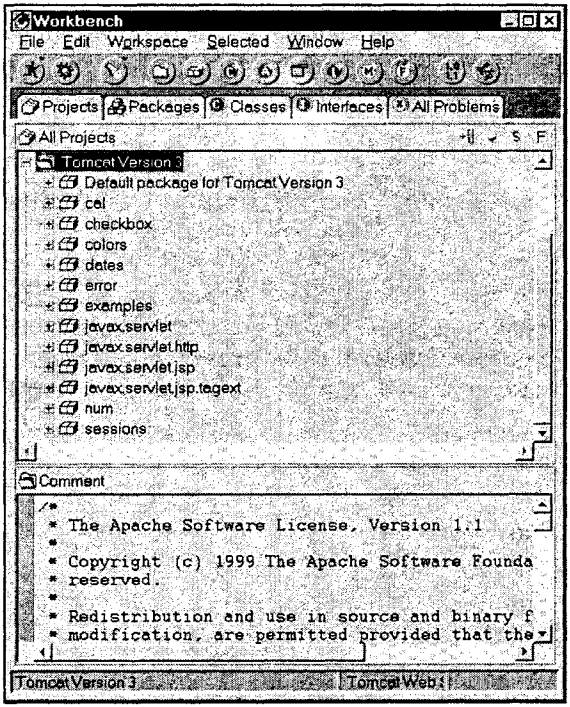


Рис. 5.4. Рабочее пространство проекта с пакетами Tomcat

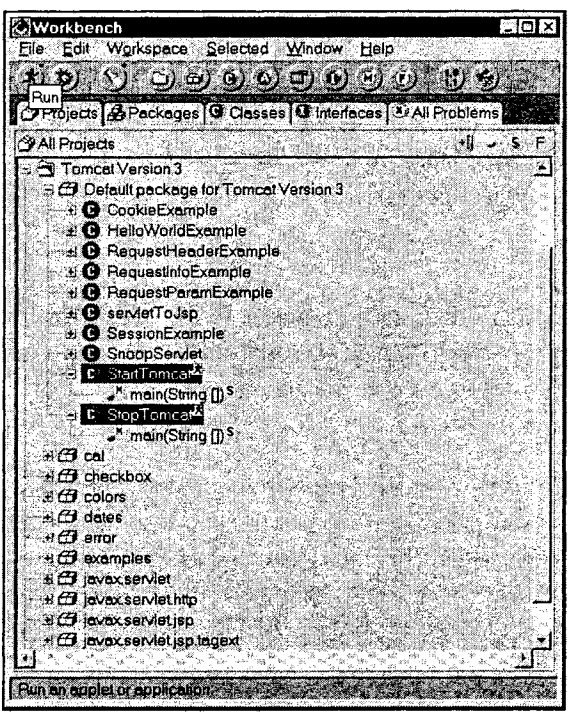


Рис. 5.5. Пуск и останов Tomcat

Отладка и тестирование

Теперь, когда Tomcat инсталлирован и запущен для работы, можно попытаться "поиграть" с различными программами, которые поставляются вместе с ним, например, с сервлетом "Hello World!". Конечно, можно сразу начать программировать сервлеты и JSP-страницы для электронного бизнеса, но не лучше ли сначала разобраться с основополагающими моментами? Тем более, что в разгрузочном пакете Tomcat есть много полезных вещей для лучшего понимания новых спецификаций Servlet 2.2 и новой версии JSP.

1. Если Tomcat Web-сервер еще не выполняется, запустите его. Вы увидите в окне VisualAge (рис. 5.6) степень готовности сервера и его параметры, доступные через конфигурационные файлы в формате XML. Советы по оптимальной настройке можно, как и всегда, найти в файле Readme.
2. Следующим шагом нужно ввести в браузере адрес **http://localhost:8080**. По этому URL обычно располагается домашняя страница Tomcat (рис. 5.7).
3. Чтобы выполнить сервлет HelloWorldExample, вы должны в браузере указать его адрес: **http://localhost:8080/servlet/HelloWorldExample**, поскольку этот сервлет доступен через рабочее пространство VisualAge в пакете по умолчанию и является частью примера Web-приложения, демонстрирующего возможности сервлетов и JSP. Список доступных приложений можно увидеть в специальном окне (рис. 5.8).
4. Щелкните по ссылке **Hello World Execute**. Сервлет запустится на выполнение и при этом на консоли появятся трассирующие сообщения (рис. 5.9).
5. На экране вашего браузера появится сообщение: "Hello World!" (рис. 5.10).

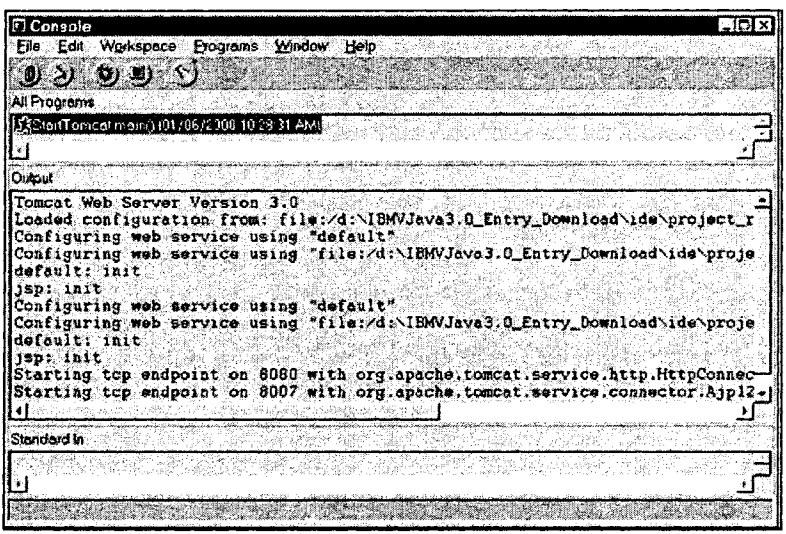


Рис. 5.6. Информация на консоли при старте Tomcat

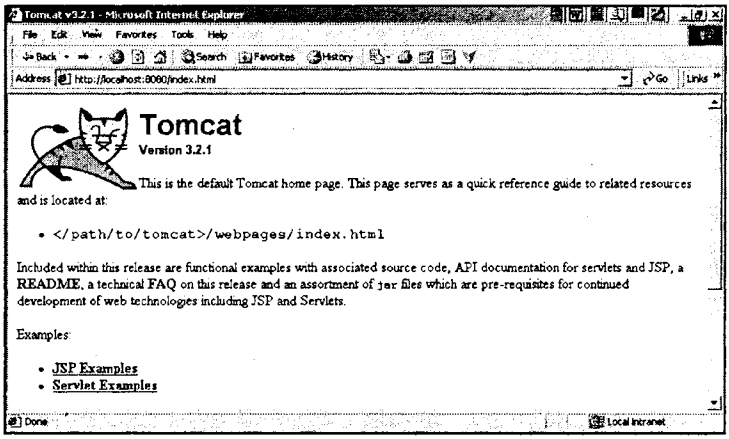


Рис. 5.7. Домашняя страница Tomcat

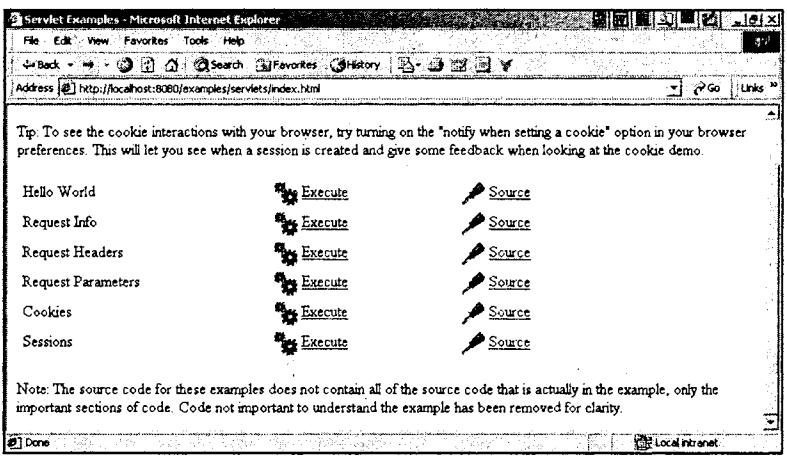


Рис. 5.8. Список доступных приложений

На этом шаге можно убедиться в том, что VisualAge for Java может выполнять Web-сервер из собственной оболочки и реализовать HTML, графические образы и сервлеты. В принципе, это очень удобно, потому что не нужно прерываться каждый раз при внесении в программу некоторых изменений и можно пользоваться встроенным отладчиком, чтобы в пошаговом режиме просматривать исполняемый код Java. Для демонстрации этой возможности будем перезагружать сервлет, после установки точек прерывания в методе сервлета `doGet`, и затем возобновлять выполнение.

Выполним эти шаги:

1. Локализуем метод `doGet` в сервлете HelloWorldExample в пакете Tomcat Version 3. Установим точку прерывания на операторе `out.println()` (рис. 5.11).

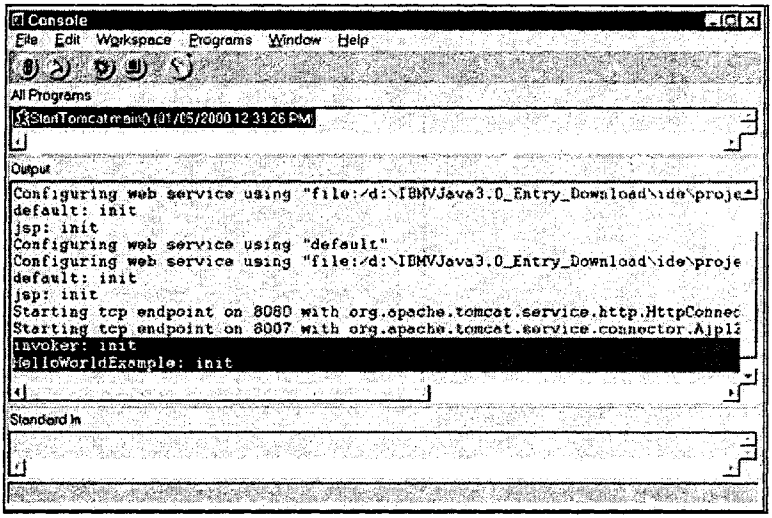


Рис. 5.9. Трассирующие сообщения

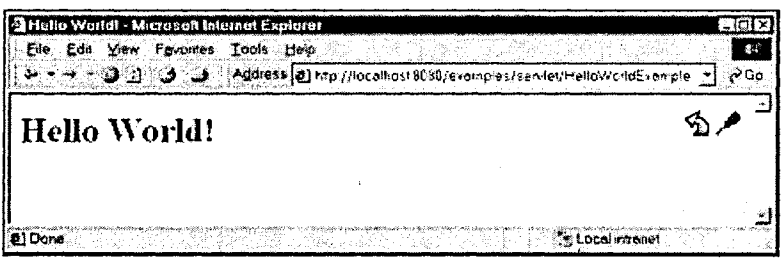


Рис. 5.10. Результат выполнения сервлета

2. Возвратимся в браузер и выполним действие **Refresh** или **Reload**. Сдвинем немного окно браузера, чтобы видеть окно отладчика. В момент достижения точки прерывания, мы вновь окажемся в отладчике и получим доступ ко всем ресурсам приложения. Рис. 5.12 показывает окно отладчика, в котором просматривается содержимое поля `contentType` в текущем объекте `response`.

3. Можно продолжить работу, выполнив операцию **Resume** (кнопка с треугольником на панели), после чего опять напечатается "Hello World!" в окне браузера. Теперь сделаем изменение в коде:

   ```
   out.println("<body> bgcolor=\"white\">");
   ```

   на:

   ```
   out.println("<body> bgcolor=\"red\">");
   ```

   чтобы изменить цвет фона с белого на красный (рис. 5.13).

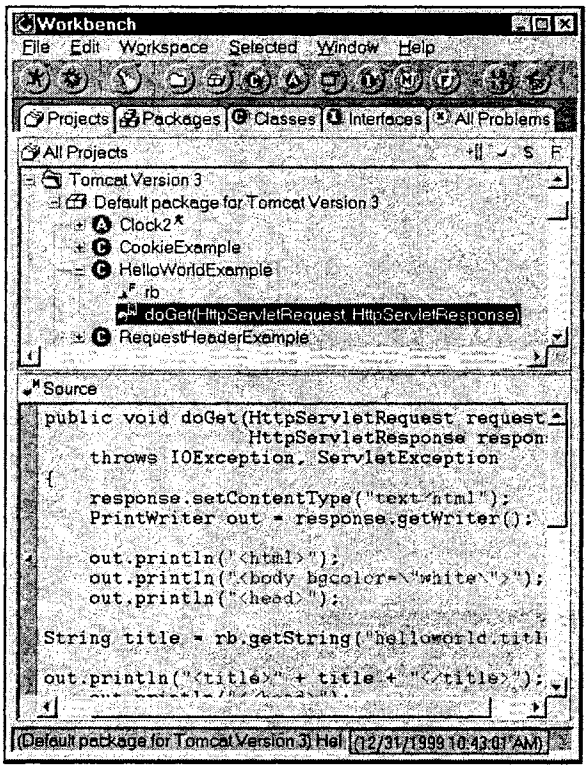


Рис. 5.11. Установка точки прерывания

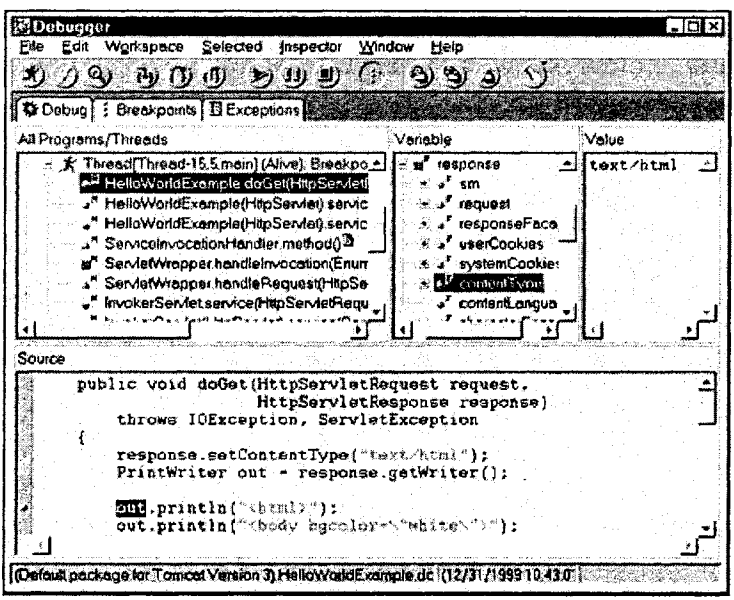


Рис. 5.12. Окно отладчика

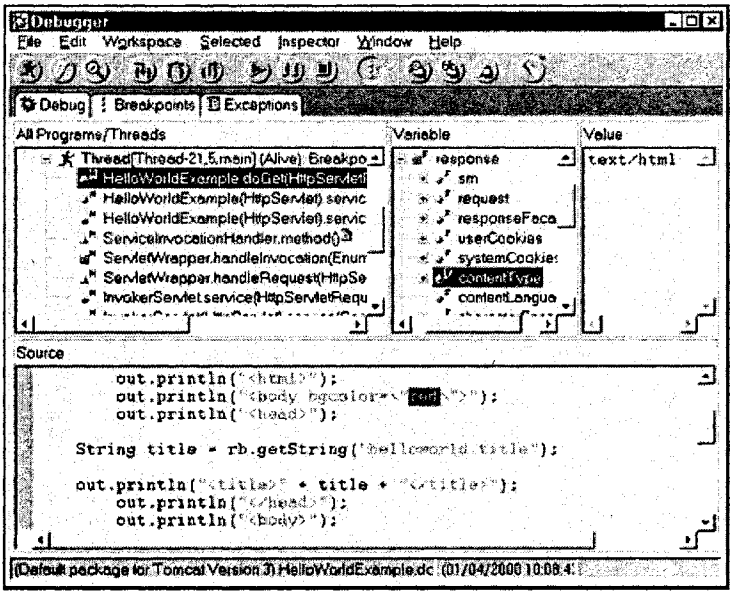


Рис. 5.13. Демонстрация изменения в коде программы

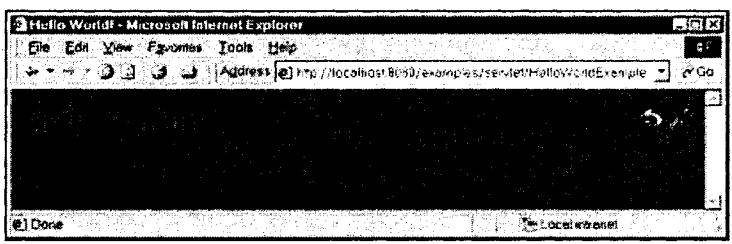


Рис. 5.14. Результат выполнения программы после корректировки

4. Затем, щелкните правой кнопкой мыши где-нибудь в исходной области окна и выберите команду **Save** из контекстного меню. В этой точке ваши изменения в исходном коде сервлета актуализируются при помощи быстрого связывания. Повторно нажмите кнопку **Resume**. Если у вас еще имеются контрольные точки, управление будет передано на следующую, если нет — сервлет выполнится до конца, и вы увидете в браузере очередной экран (рис. 5.14).

В дополнение к отладке сервлетов, VisualAge for Java поддерживает отладку JSP-страниц.

Отладка компонентов JavaBean, вызываемых из сервлетов и JSP-файлов

Чтобы дать возможность опробовать в VisualAge for Java инструменты отладки и выполнения JSP-кода, мы вновь продемонстрируем один из примеров, который включен в дистрибутив Tomcat.

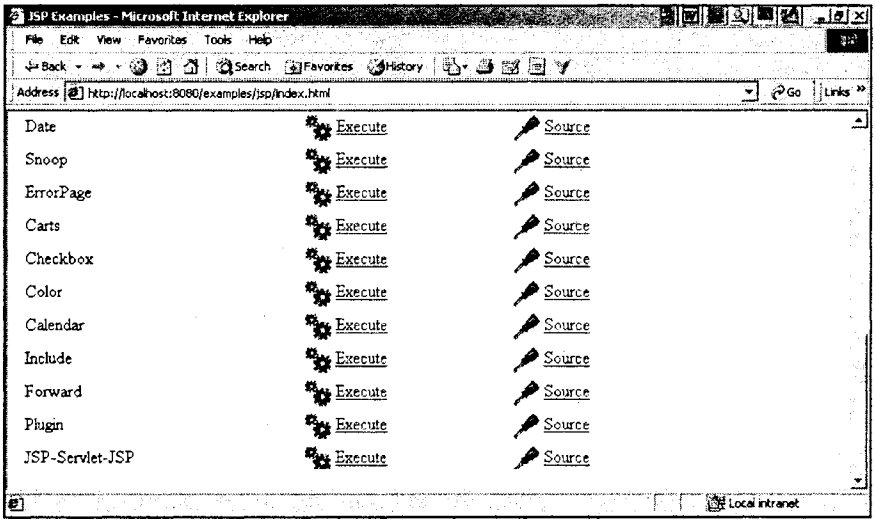


Рис. 5.15. Список примеров JSP

Выполните следующие шаги:

1. Убедитесь, что Tomcat-сервер стартовал и выполняется.
2. Перейдите в браузере на домашнюю страницу **http://localhost:8080 Tomcat**.
3. Щелкните ссылку **JSP-Servlet-JSP Execute**. Вы увидите страницу, изображенную на рис. 5.15.

   Исходный модуль date.jsp использует Bean dates.JspCalendar, для того чтобы обеспечить выполнение на сервере некоторой программной логики. Этот Bean, в принципе, можно отладить прямо из окна браузера. (Обратите внимание на код, отмеченный полужирным шрифтом.)

```
<html>
  <!-- Copyright (c) 1999 The Apache Software Foundation. All rights
reserved. -->
  <%@ page session="false"%>
  <body bgcolor="white">
    <jsp:useBean id='clock' scope='page' class='dates.JspCalendar'
     type="dates.JspCalendar" />
    <font size=4>
      <ul>
        <li> Day of month: is <jsp:getProperty name="clock" prop-
erty="dayOfMonth"/>
        <li> Year: is <jsp:getProperty name="clock" property="year"/>
        <li> Month: is <jsp:getProperty name="clock" property="month"/>
        <li> Time: is <jsp:getProperty name="clock" property="time"/>
```

```
        <li> Date: is <jsp:getProperty name="clock" property="date"/>
        <li> Day: is <jsp:getProperty name="clock" property="day"/>
        <li> Day Of Year: is <jsp:getProperty name="clock" prop-
erty="dayOfYear"/>
        <li> Week Of Year: is <jsp:getProperty name="clock" prop-
erty="weekOfYear"/>
        <li> era: is <jsp:getProperty name="clock" property="era"/>
        <li> DST Offset: is <jsp:getProperty name="clock" prop-
erty="dSTOffset"/>
        <li> Zone Offset: is <jsp:getProperty name="clock" prop-
erty="zoneOffset"/>
      </ul>
    </font>
  </body>
</html>
```

4. Щелкните кнопку **Execute** для выполнения файла date.jsp. После некоторой задержки, во время которой Tomcat компилирует, загружает и выполняет JSP-код с динамической генерацией сервлета "worker", в браузере отображается форма. Все последующие запросы к этому JSP (если вы только не изменяете его исходный код) будут производиться намного быстрее, потому что сервлет уже скомпилирован, загружен и, следовательно, готов обслуживать запросы от клиентов. Результат выполнения JSP-сценария представлен на рис. 5.16.

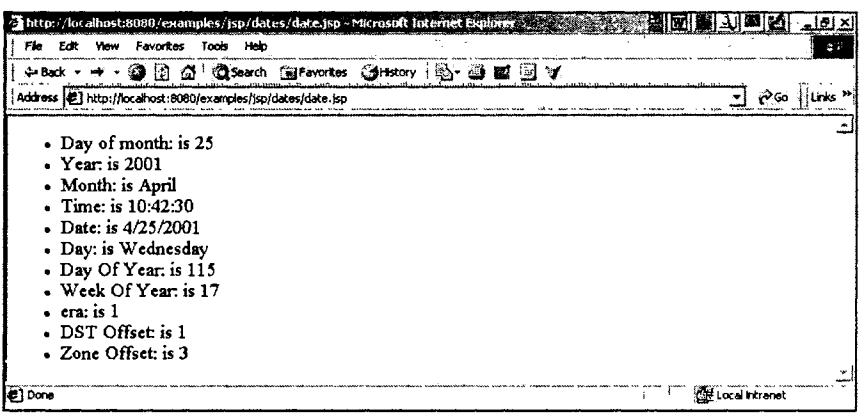


Рис. 5.16. Результат выполнения JSP-сценария

5. Чтобы отлаживать Bean dates.JspCalendar из браузера, установите точку прерывания в одном из методов этого компонента. К примеру, установим точку прерывания в методе `getDay()` (рис. 5.17). Сам Bean — это пакет, называющийся `Dates` и являющийся частью проекта Tomcat Version 3. Вы можете заметить, что в компоненте имеется выполняемый метод `main`.

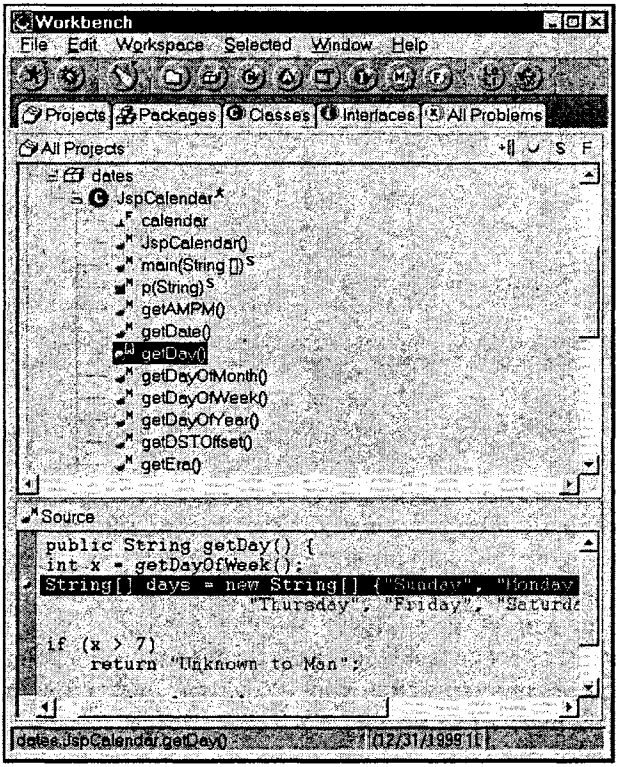


Рис. 5.17. Установка точки прерывания для отладки компонента Bean

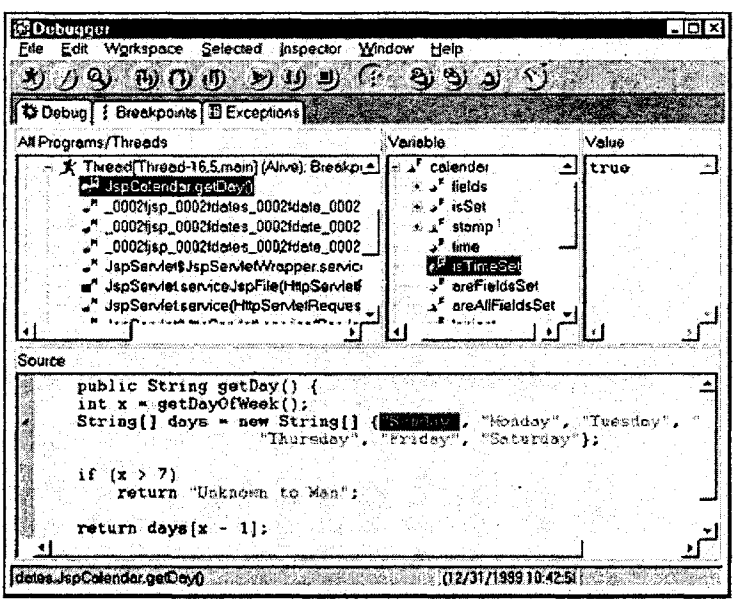


Рис. 5.18. Окно отладчика для компонента Bean

6. Перезагрузим страницу из браузера. Встроенный отладчик должен при этом показать свое окно (рис. 5.18), когда программа достигнет точки прерывания. Вы можете изменить код по своему усмотрению и продолжить выполнение без всякой перекомпоновки выполняемого компонента, существующего в виде Bean.

Отладка JSP-файлов

Как часть JSP-модели, JSP-модуль компилируется в некий объект, который должен выполнять протокол сервлета. В настоящее время большинство реализаций, в т. ч. Tomcat, порождают HTTP-сервлеты, которые включают обе части: статический HTML и связанную с ним динамическую программу на Java. Когда происходит нечто непредвиденное, было бы неплохо посмотреть в деталях этот сгенерированный сервлет. Встроенный в VisualAge отладчик как нельзя лучше подходит для этой цели. Чтобы отлаживать рабочий сервлет, необходимо:

1. Стартовать сервер Tomcat, если он еще не запущен.
2. Вновь обратиться к файлу date.jsp. Если вы еще не перезапускали Tomcat, у вас должны остаться сгенерированные Java-файлы в каталоге \work \8080%2Fexamples (рис. 5.19). Имена этих файлов могут быть подобны следующим:

```
_0002fjsp_0002fdates_0002fdate_0002ejsp_0002fjsp_0002fdates_0002fdate.
class
_0002fjsp_0002fdates_0002fdate_0002ejsp_0002fjsp_0002fdates_0002fdate_
jsp_0.java
```

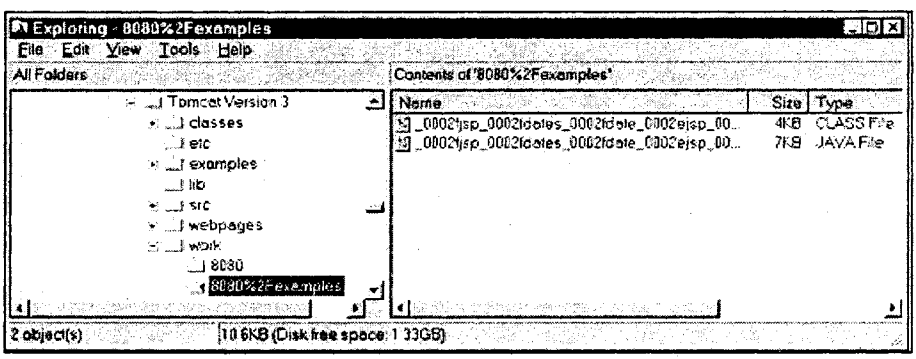


Рис. 5.19. Окно с именами файлов

3. Откройте отладчик и выберите последовательно команды **Methods, External .class file breakpoints** (рис. 5.20).
4. Используйте диалог **Set breakpoints in external .class file** для просмотра сгенерированного class-файла, который вы хотите отладить.

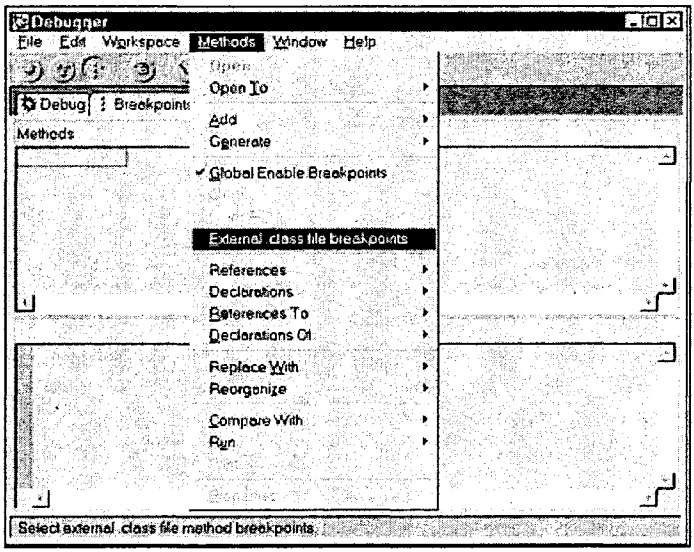
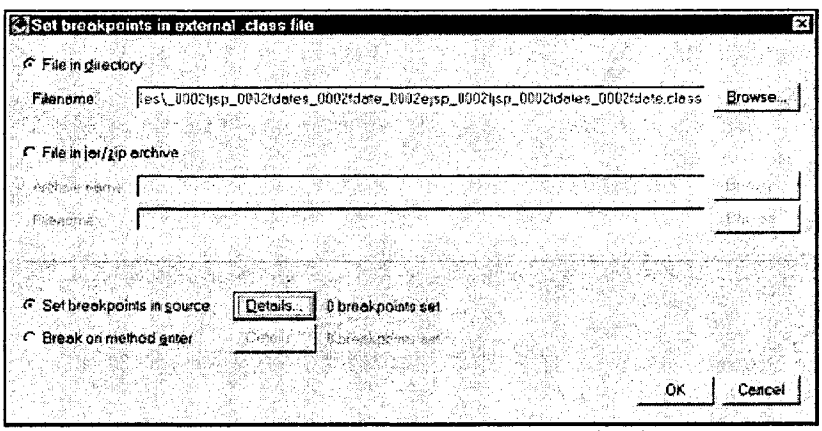


Рис. 5.20. Выбор нужного метода



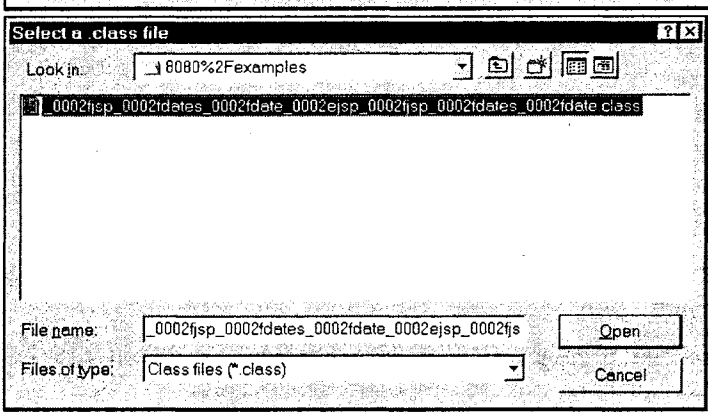


Рис. 5.21. Открытие необходимого файла

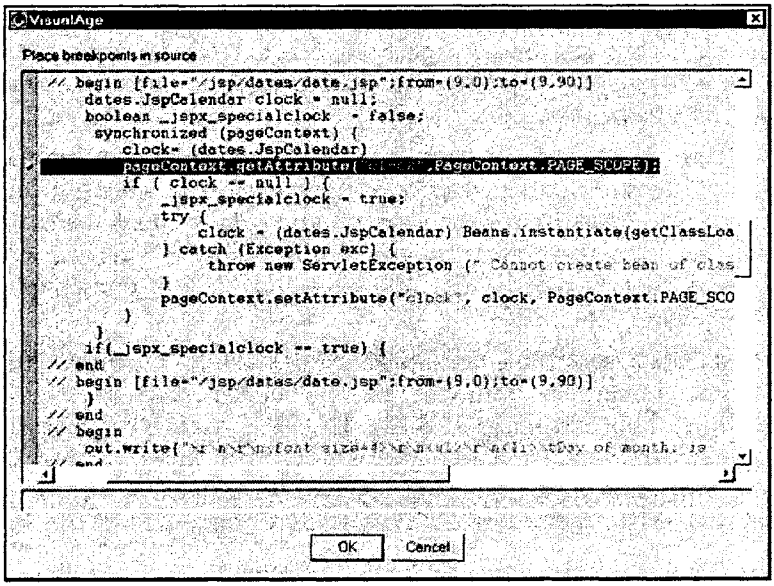


Рис. 5.22. Установка точки прерывания для отладки JSP-файла

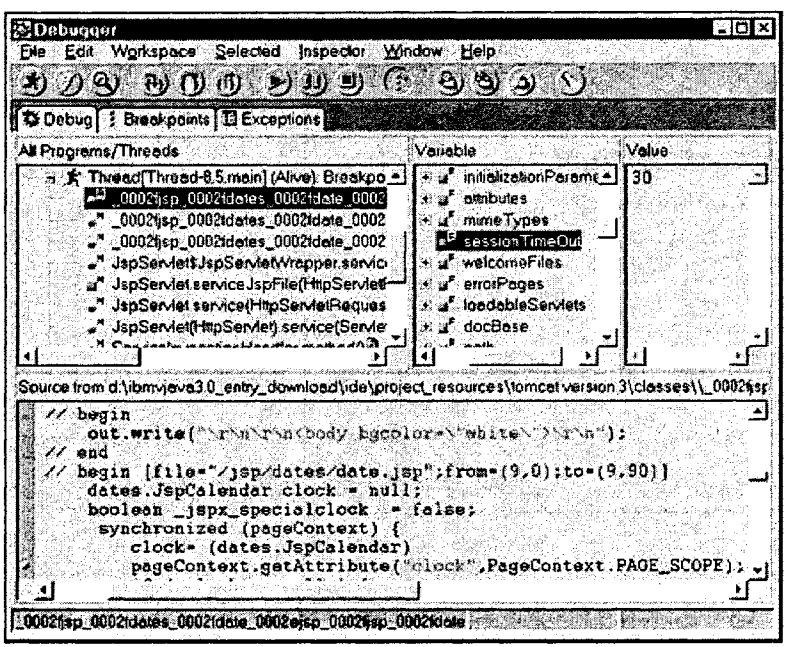


Рис. 5.23. Окно отладчика в режиме пошагового выполнения

К примеру, отметьте переключатель **File in directory** и затем откройте файл с расширением .class для просмотра (рис. 5.21). По умолчанию, он должен быть в каталоге IBMVJavaRoot\ide\project_resources\Tomcat Version 3\work\8080%2Fexamples.

5. На следующем шаге вернитесь в диалоговое окно **Set breakpoints in external .class file**, выберите переключатель **Set breakpoints in source** и нажмите кнопку **Details**. Появится диалоговое окно **Place breakpoints in source**, в котором вы можете установить точку прерывания. Укажите ее, например, в строке `pageContext.getAttribute method` в JSP-сгенерированном рабочем сервлете (рис. 5.22).
6. Вновь перезапустите date.jsp, чтобы остановиться в точке прерывания. Вы можете продолжить пошаговое выполнение, наблюдая, как изменяют значения переменные программы (рис. 5.23). Имеется только одно ограничение — вы не можете вносить изменения в исходный код программы. Чтобы, все-таки, это сделать, вы должны выполнять отладчик в классе, загружаемом непосредственно из Workspace, как описано ранее.
7. Когда вы закончите отлаживать внешний сгенерированный JSP-код, необходимо очистить все контрольные точки (рис. 5.24).

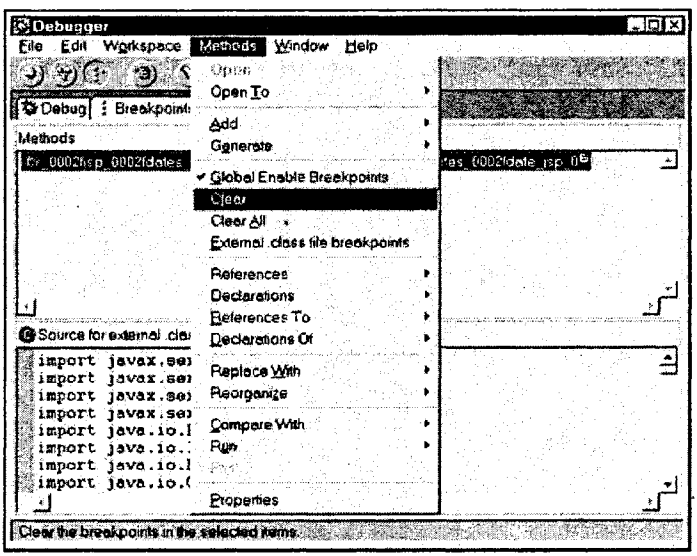


Рис. 5.24. Очистка контрольных точек

Некоторые проблемы

Если вдруг случится ошибка в браузере, подобная такой:

```
Error: 404
JSP file "work\8080%2Fexamples\_0002fjsp_0002fdates_0002fdate_
0002ejsp_0002fjsp_0002fdates_0002fdate_jsp_0.java" not found
```

завершите выполнение Tomcat-сервера и полностью удалите каталог entire (со всеми его подкаталогами), указанный в секции workDir файла server.xml. По умолчанию, этот каталог называется <IBMVJavaRoot>\ide\project_resources\Tomcat Version 3\work. В нем сохраняются временные файлы, которые удаляются после того, как сервер завершит свою работу.

Не забывайте очищать все точки прерывания, чтобы правильно впоследствии запустить сервер.

Имеются некоторые ошибки в самом отладчике VisualAge 3.0, которые, вероятно, будут устранены с выходом новой версии продукта.

Поскольку Tomcat был выпущен сравнительно недавно, никто не может поручиться за его бесспорную устойчивость к сбоям. Так что вполне допустим случай, когда вы подумаете, что имеется проблема с VisualAge, хотя на самом деле, она может быть вызвана Tomcat.

Сервер приложений IBM WebSphere 3.0

Из всех серверов приложений WebSphere 3.0 — сервер Web-приложений корпорации IBM — хорош тем, что он наиболее полно реализует все современные стандарты программирования для Интернета и усиленно поддерживает Java и XML. Кроме того, он содержит прекрасные средства администрирования (рис. 5.25).

Благодаря этим усовершенствованиям WebSphere среди серверов Web-приложений выходит на одно из первых мест по степени полноты поддержки стандартов. WebSphere имеет средства поддержки разработчика — WebSphere Studio 3.0. С помощью этого пакета можно быстро создавать Web-приложения, которые затем легко развивать. В частности, поддержка технологии EJB (Enterprise JavaBeans) автоматически обеспечивает многозадачность, безопасность, транзакции с БД и управление текущим состоянием — в других системах программисту приходится добиваться этого самостоятельно.

Кроме того, WebSphere позволяет осуществлять разработку Web-приложений с сервлетами — написанными на Java серверными компонентами, которые проще в развертывании, чем компоненты EJB или Java Server Pages.

Тем не менее, поддерживаемая WebSphere JSP-технология обеспечивает наивысшие простоту и скорость разработки — по этим показателям она сопоставима с такими системами, как сервер ColdFusion корпорации Allaire или Active Server Pages в Internet Information Server корпорации Microsoft.

Совместная модель EJB и Java Server Pages — особенно сильная сторона WebSphere 3.0.

Уже в WebSphere 2.0 появились библиотеки поддержки расширяемого языка разметки XML (Extensible Markup Language) — для чтения и записи XML-

документов. Имеется встроенная поддержка трех основанных на XML форматов. Формат данных XML может применяться для обмена информацией между разнородными системами, используемыми в вычислительной системе предприятия и его партнеров. Расширенный набор средств Web-администрирования WebSphere (см. рис. 5.23) позволяет следить за функционированием сервера и полностью контролировать его поведение. Контроль за функционированием множественных серверов и управление их работой могут осуществляться с помощью системы сетевого администрирования Tivoli TME-10, для которой IBM выпускает дополнительные модули. Разработчики IBM ввели в WebSphere возможность интерактивной дистанционной отладки страниц и бизнес-логики приложений из внешнего инструмента разработки на Java. В частности, IBM встроила в среду разработки VisualAge for Java Enterprise Edition, поддерживающую технологию EJB, мастера для генерации развертывания кода на серверах WebSphere.

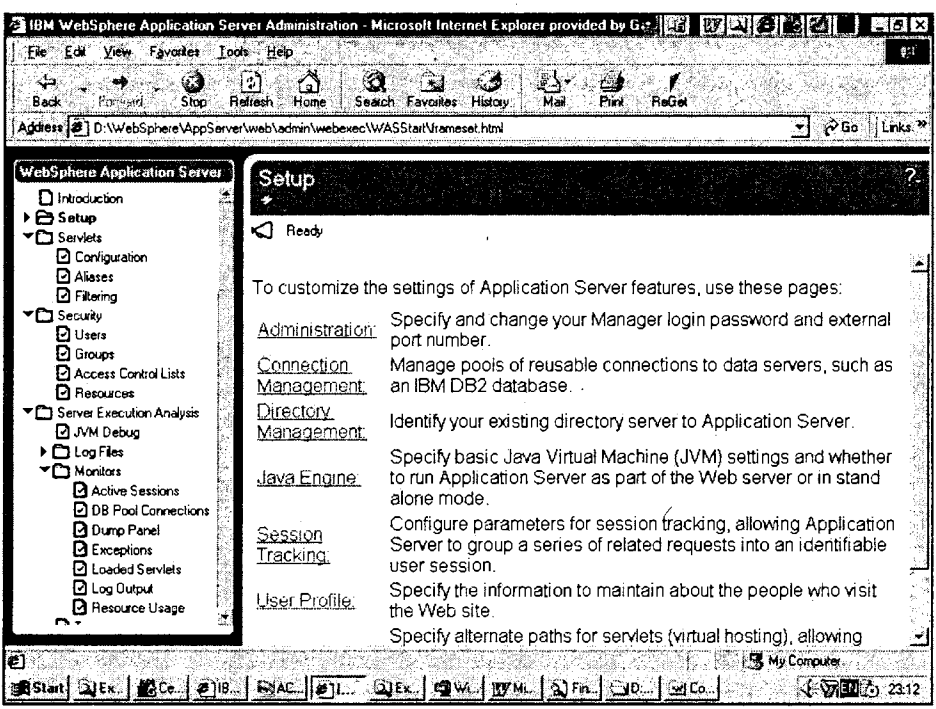


Рис. 5.25. Из браузера можно контролировать использование ресурсов в WebSphere и время его отклика

В WebSphere 2.0 повышена избирательность настроек подсистемы обеспечения безопасности. Сервер позволяет определять права доступа пользователей и групп пользователей к индивидуальным HTML-файлам и Java-сервлетам.

В дополнение к интеграции со средствами обеспечения безопасности ОС Solaris и Windows NT, WebSphere поддерживает теперь аутентификацию пользователей с помощью внешнего сервера на базе протокола Lightweight Directory Access Protocol (LDAP, упрощенный протокол доступа к каталогам). Для разработчиков Web-приложений с использованием WebSphere имеется специальное программное средство WebSphere Studio 3.0.

WebSphere Studio 3.0

Этот продукт по большому счету сильно напоминает другие известные продукты для проектирования Web-сайтов, такие, например, как FrontPage или Macromedia Dreamweaver, но от них он отличается тем, что во всем богатстве использует почти все современные Java-технологии, такие как EJB, JSP и др., особенно те, которые бесшовно встраивают работу с SQL-серверами в единое технологическое пространство и позволяют использовать единые технологические инструменты. Чего только нет в WebSphere Studio?

- **Studio workbench.** Среда проектирования с богатыми возможностями управления Web-объектами, которая позволяет их группировать, копировать, перемещать, удалять, редактировать, связывать, публиковать.
- **JavaBean, Database и SQL-мастера.** С помощью этих мастеров можно без особого труда создавать готовые компоненты и поручать им работу, результатами которой можно впоследствии воспользоваться на сайте. Это может быть работа по извлечению данных из баз и построение результирующих и аналитических таблиц, поисковые роботы, системы управления информационным содержанием и т. д. Любой из перечисленных компонентов может представлять собою черный ящик, обладающий массой возможностей, которым чрезвычайно легко и даже приятно управлять без всякой необходимости знать встроенные в него механизмы и протоколы, подобные многочисленным Java-спецификациям или синтаксису языка SQL.
- **JAR wizard.** Этот мастер преобразует любой Java-класс в стандартный JavaBean.
- **Import wizard.** С помощью этого мастера можно импортировать любой существующий сайт в новый проект и перенастроить его в соответствии с требованиями новейших технологий.
- **Page Designer с WebArt Designer и AnimatedGif Designer.** Эти средства являются мощными инструментами в руках художника-дизайнера и удовлетворяют любой, даже самый изысканный художественный вкус. Имеются встроенные многочисленные палитры и художественные элементы.
- **Applet Designer.** Мастер позволяет автоматизировать рутинные процедуры создания апплетов и встраивания их в Web-страницы.

Таким образом, WebSphere Studio представляет собой мощное средство проектирования, многозвенных распределенных Web-приложений, а WebSphere

Server является идеальной машиной для исполнения этих приложений практически на любом из существующих HTTP-серверов, например, как MS IIS, Apache, Domino Server и др.

WebSphere Application Server существует в трех вариантах:

- WebSphere Application Server Standard Edition;
- WebSphere Application Server Advanced Edition;
- WebSphere Application Server Enterprise Edition.

WebSphere Application Server Standard Edition

IBM WebSphere Application Server v2.0 Standard Edition является идеальной платформой для открытых Java-технологий. Центральное звено продукта — это многофункциональная машина сервлетов для запуска и управления Java-приложениями, выполняющимися на сервере.

Ключевые особенности:

- многофункциональный сервер HTTP от IBM (основанный на технологии сервера HTTP Apache);
- поддержка Lotus Domino версии 5.0;
- Tivoli-совместимые модули (дополнительно предлагается вариант их загрузки с Web), которые могут управляться посредством системы Tivoli;
- улучшенное администрирование приложений и серверов;
- сервисы структуры документов XML;
- полная поддержка Java Servlet, JavaBeans, Java Server Pages (JSP), Extensible Markup Language (XML) и Extensible Style Language (XSL);
- Java Servlet 2.1 API;
- динамический доступ к базам данных с использованием Java Database Connectivity (JDBC), включая доступ к DB2 Universal Database, Oracle и Microsoft SQL Server;
- улучшенная интеграция с IBM VisualAge for Java, позволяющая отлаживать код Java и JSP локально или удаленно внутри, либо WebSphere Application Server, либо WebSphere Studio.

Дополнительные возможности:

- отслеживание сеансов;
- отслеживание места на сайте, которое посетитель просматривает в данный момент;
- персонализация и управление профилями пользователей;
- доступность информации для использования в Web-приложениях;

- динамическое содержимое страниц;
- возможность для посетителей обмениваться сообщениями друг с другом;
- защищенность;
- фиксация страницы, с которой посетитель попал на сайт;
- ассоциация идентификации пользователя с сеансом;
- возможность работы с детальной информацией о пользователях;
- поддержка замены переменных на Web-страницах;
- возможность для администраторов динамически публиковать бюллетени, отображаемые на всех страницах сайта;
- монитор активности сайта — динамический обзор в реальном времени.

WebSphere Application Server Advanced Edition

IBM WebSphere Application Server v2.0 Advanced Edition сочетает управление контролем и переносимостью серверных бизнес-приложений с производительностью и управляемостью модели Sun Enterprise JavaBean, что делает этот сервер полноценной, основанной на Java, платформой для Web-приложений.

Ключевые особенности:

- полнофункциональный сервер HTTP от IBM (основан на технологии HTTP-сервера Apache) с дополнительными функциями защиты на базе SSL и производительности;
- поддержка Lotus Domino версии 5.0;
- Tivoli-совместимые модули (дополнительно предлагается вариант их загрузки с Web), которые могут управляться посредством системы Tivoli;
- улучшенный интерфейс для администрирования приложений и серверов;
- сервисы структуры документов XML;
- дополнительный акцент на EJB Engine (машине EJB) с поддержкой CORBA, усовершенствованной для обеспечения bean- и контейнер-управляемой устойчивости;
- Enterprise JavaBeans Server, обеспечивающий управление и мониторинг транзакциями для реляционных баз данных, основанный на компонентах Enterprise JavaBeans и CORBA.

Компоненты:

- EJB Engine;
- машина сервлетов;
- менеджер внедрения;
- сервисы соединений;

- сервисы приложений;
- Web-сервер Apache;
- сервер баз данных (для устойчивого хранилища EJB).

Серверы, поддерживаемые WebSphere Application Server v2.0 Advanced Edition:

- Apache Server Version 1.3.2 for AIX, Sun Solaris и Windows NT;
- IBM HTTP Server V1.3.3 (на основе Apache) для AIX, Sun Solaris и Windows NT;
- Netscape Enterprise Server Version 3.01 и 3.51 для AIX, Sun Solaris и Windows NT;
- Netscape FastTrack Server Version 3.01 для AIX, Sun Solaris и Windows NT;
- Microsoft Internet Information Server Version 3.x и 4.0 для Windows NT;
- Lotus Domino V5 для AIX и Windows NT;
- Lotus Domino Go Webserver V4.6.2.5 для AIX, Windows NT, OS/2 (для серверов) и Sun Solaris.

WebSphere Application Server Enterprise Edition

IBM WebSphere Application Server v2.0 Enterprise Edition включает все усовершенствования для Advanced Edition и предлагает новые мощные решения, обеспечивающие выполнение приложений в сфере электронного бизнеса для крупных корпораций. В Enterprise Edition скомбинированы возможности TXSeries — среды для работы с транзакционными приложениями и интеграционные возможности полностью распределенных объектов и бизнес-процессов Component Broker.

Enterprise Edition включает все то же, что и Advanced Edition, а именно:

- полнофункциональный сервер HTTP от IBM (основан на технологии HTTP-сервера Apache) с дополнительными функциями защиты на базе SSL и производительности;
- поддержку Lotus Domino версии 5.0;
- Tivoli-совместимые модули (дополнительно предлагается вариант их загрузки с Web), которые могут управляться посредством системы Tivoli;
- улучшенный интерфейс для администрирования приложений и серверов;
- сервисы структуры документов XML;
- поддержку CORBA, усовершенствованную для обеспечения bean- и контейнер-управляемой устойчивости;
- Enterprise JavaBeans Server, обеспечивающий управление и мониторинг транзакциями для реляционных баз данных, основанный на компонентах Enterprise JavaBeans и CORBA;

- возможности по обработке транзакций — поддержкой TXSeries, среды мирового класса для транзакционных приложений от IBM — с поддержкой Component Broker, с его интеграционными возможностями полностью распределенных объектов и бизнес-процессов.

Компоненты:

- IBM WebSphere Application Server Advanced Edition V3.5;
- IBM Component Broker V3.5 (server);
- IBM TXSeries;
- IBM CICS Transaction Gateway, V3.0.2;
- IBM CICS Universal Clients, V3.1;
- IBM MQSeries, V5.1;
- IBM DCE for AIX, V3.1;
- IBM DCE for Windows NT, V2.2;
- IBM/Transarc DCE for Solaris, V2.0.

Элементы WebSphere-приложений

Сервер приложений WebSphere является частью генеральной стратегии фирмы IBM для завоевания ею части глобального информационного пространства и основой для построения систем электронной коммерции (рис. 5.26).

Функциональное ядро WebSphere образуют следующие базовые технологии:

- Java-сервлеты
- Java Server Pages
- Enterprise JavaBeans

Java-сервлеты

Сервлеты — это просто маленькие Java-программы, которые, в отличие от апплетов, не требуют загрузки клиентом на его собственную машину, а выполняются прямо на сервере приложений (Web-сервере). Сервлеты легко транспортируются между различными платформами и архитектурами и различными Web-серверами. Обычно они взаимодействуют со специальной сервлет-машиной, которая выполняется на сервере приложений. Сервлеты посылают к ней запросы и получают ответы, которые инкапсулируются в объекты.

Сервлет загружается сервером приложений и вызывается им каждый раз при получении очередного клиентского запроса, а это означает, что сервлеты экономят ресурсы системы.

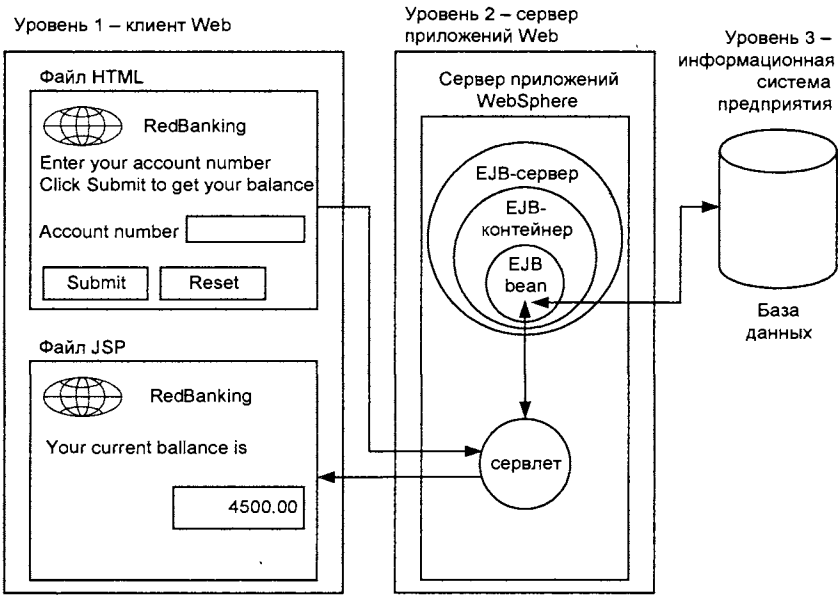


Рис. 5.26. Взаимосвязанные компоненты WebSphere образуют платформу для построения систем электронной коммерции

Сервлеты расширяют класс `javax.servlet.http.HttpServlet`. Наиболее типичными методами, которые реализуют сервлеты, являются следующие:

- `init()` — вызывается, когда сервлет загружается в первый раз, но он не вызывается при повторном и последующих обращениях к сервлету;
- `doGet (HttpServletRequest req, HttpServletResponse resp)` — вызывается, если сервлет получает управление посредством GET-запроса, а так же, когда сервлет получает управление путем ввода URL в браузере;
- `doPost (HttpServletRequest req, HttpServletResponse resp)` — вызывается, если сервлет активизируется по POST-запросу (обычно этот запрос посылается при использовании HTML-форм).

Блок параметров `HttpServletRequest` содержит атрибуты запроса для сервлета. Вы можете использовать `HttpServletResponse` для генерации результатов ответа, посылаемого к клиенту, после того как сервлет завершит порученную ему работу.

Полная документация о Java Servlet API доступна по адресу: **http://java.sun.com/products/servlet/2.2/javadoc/index.html**.

Java Server Pages

Java Server Pages (JSP) — это наиболее простой способ комбинирования Java-кода и HTML, что позволяет разделить манипуляции с данными и спо-

собы их представления на странице. В Web-приложении блоки результатов, посылаемые к клиенту, часто представляют собой набор статичных страниц и динамически генерируемых данных, которые могут появляться, исчезать, а потом снова выскакивать в различных частях этих страниц. В этой ситуации нет ничего легче и удобнее, чем использовать JSP, вместо сервлетов, которые конструируют класс `HttpServletResponse` для Web-страницы.

JSP — это HTML-файлы, содержащие добавочные теги. WebSphere поддерживает версии 0.91 и 1.0 JSP-спецификации и добавляет некоторые свои теги. Вы можете размещать Java-код между некоторыми JSP-тегами. Наиболее важные теги, которые позволяют это сделать, таковы.

- Импорт операторов

```
<%@ language="java" import="java.sql.*" %>
```

- Объявление переменных и методов

```
<%! int iStatus = 0; %>
```

- Выражения

```
<%= request.getParameter("message") %>
```

- Скриптлеты (содержат кодовые фрагменты коротких Java-программ)

```
<% for (int iCounter = 0; iCounter < vejbAccounts.size(); iCounter ++) {
  AccountDataBean ejbAccount =
  (AccountDataBean) vejbAccounts.elementAt(iCounter);
  int iAccountID = ejbAccount.getID();
}%>
```

В добавление к этому, вы можете включать HTML-страницы, сервлеты и EJB в JSP. Подробное описание данных тегов можно найти в предыдущей главе, в разделе, посвященном JSP, или на сайте **http://java.sun.com /products/jsp/tags/11/tags11.html**.

Когда новый или модифицированный JSP вызывается впервые, WebSphere генерирует временный сервлет. Этот сервлет будет вызываться каждый раз, когда браузер будет обращаться к JSP (рис. 5.27).

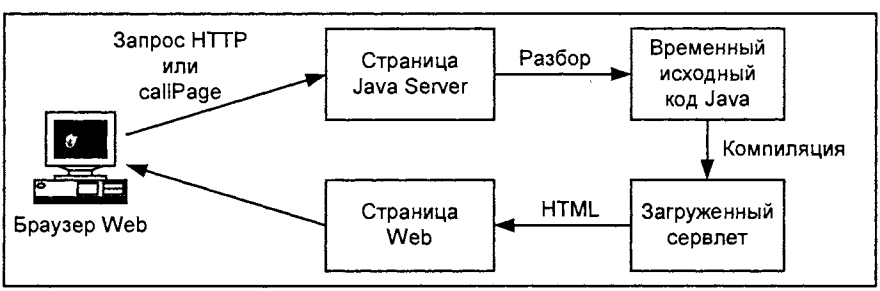


Рис. 5.27. Обработка JSP в WebSphere

В некотором смысле JSP очень похожи на ASP (Active Server Pages) фирмы Microsoft. Отличие, однако, заключается в том, что здесь используется Java, а в ASP — Visual Basic. В будущем IBM обещает встроить в JSP и LotusScript для более тесной интеграции Domino и WebSphere.

Enterprise JavaBeans

Enterprise JavaBeans — это торговая марка Sun для ее EJB-архитектуры (или компонентной модели — component model).

EJB инсталлируются на сервере, и доступ к ним осуществляется удаленно со стороны клиентов.

EJB-клиентом может быть любая программа, которая взаимодействует с ним по протоколу Remote Method Invocation (RMI) или через Internet Inter-ORB Protocol (IIOP).

Основные шаги при использовании EJB

Существует некоторая последовательность действий, когда клиентская программа желает использовать EJB (рис. 5.28).

Эти шаги таковы:

1. Клиент запрашивает через службу поддержки имен (один из компонентов WebSphere) ссылку на интерфейс EJB Home, специфический класс EJB.
2. Служба поддержки имен (Naming Service) предоставляет клиенту такую ссылку на экземпляр EJB-класса в контейнере, в котором развернут EJB.

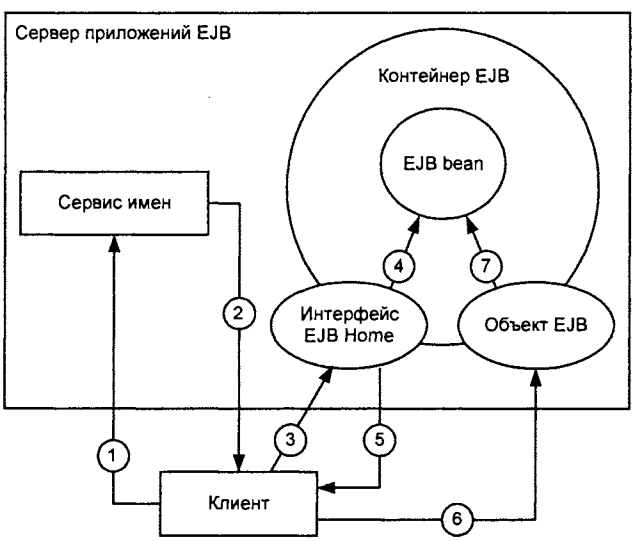


Рис. 5.28. Использование EJB в WebSphere

3. Клиент находит или создает через предоставленный ему интерфейс, соответствующий bean.
4. Формируется соответствующая объектная ссылка.
5. Интерфейс EJB Home сообщает клиенту ссылку на созданный экземпляр объекта EJB.
6. Клиент вызывает методы этого объекта, чтобы использовать встроенные в bean алгоритмы, реализующие нужные клиенту бизнес-процедуры.
7. Через объектные интерфейсы организуется взаимодействие с контейнером.

Lotus Domino и WebSphere

Хотя Lotus Domino Server 5.x сам по себе является самодостаточным продуктом, имеющим все нужные средства для разработки первоклассных Web-приложений и довольно трудно убедить многотысячную армию программистов Lotus в необходимости иметь для них еще какие-то дополнительные средства для разработки более изощренных Web-приложений (Domino обладает богатым инструментарием программирования Java-апплетов, скриплетов, сервлетов, DHTML и т. д.), — все-таки, IBM продолжает упорную работу по интеграции Domino и WebSphere. IBM, которая потратила миллионы долларов на продвижение своей инициативы в сфере электронной коммерции, планирует использовать Domino в качестве ключевого служебного компонента основного Web-сервера и, одновременно, среды разработки Web-приложений — WebSphere.

В результате их интеграции должна возникнуть платформа, которая не только будет поддерживать все электронные транзакции благодаря заложенным в WebSphere механизмам обработки транзакций и взаимодействия с различными серверными платформами, но и позволит компаниям общаться с их заказчиками посредством электронной почты, форумов и систем обмена срочными сообщениями.

"Интеграция поможет нарастить мощь Notes, расширить спектр ее возможностей", — утверждают руководители IBM, хотя пользователи Lotus относятся к этой идее без особого энтузиазма.

WebSphere Standard Edition уже в значительной мере интегрирован с Domino R5. Согласно новому плану, Domino предстоит "обвенчаться" еще и с WebSphere Advanced Edition, содержащим механизм обработки транзакций — основу любой системы электронной коммерции.

"Самая трудная задача — объединить серверы в одну логическую архитектуру, сохранив возможность самостоятельного применения каждого из них", — утверждают аналитики из Burton Group.

IBM уже давно сотрудничает с Iris Associates, отделением Lotus по разработке. Интеграцию с WebSphere планируется вести в терминах языка Java, служб каталогов и средств безопасности.

Какие конкретно выгоды получат программисты от этого слияния?

- Единый механизм регистрации пользователей
- Разделяемые сессии
- Поддержку языка LotusScript в Java Server Pages

Опция `<Script Language="LotusScript">` даст огромные преимущества программистам Lotus, потому что позволит использовать широкий спектр классов и методов, встроенных в Domino Server в новом программном окружении:

- общие процедуры инсталляции и администрирования;
- общие средства сервиса и обслуживания;
- единую среду разработки приложений, инструментальную среду программирования IDE;
- архитектурно-платформенную интеграцию.

В качестве примера рассмотрим конкретный вариант инсталляции и конфигурации WebSphere и Domino.

Требования для инсталляции

Аппаратное обеспечение:

- Pentium II или выше;
- 256 Мбайт RAM — абсолютный минимум, рекомендовано 512 Мбайт;
- 500 Мбайт дискового пространства, в том числе для:
  - DB2 — 180 Мбайт;
  - JDK — 30 Мбайт;
  - Domino — 500 Мбайт;
  - WebSphere — 120 Мбайт.

Программное обеспечение:

- Microsoft Windows NT v4.0 SP4 или SP5. Может использоваться либо рабочая станция (Workstation), либо сервер;
- TCP/IP-протокол с выделенным IP-адресом (или loopback-адаптер для отдельной машины).

Основные шаги

Регистрация пользователя

WebSphere Application Server, Domino и DB2 должны выполняться под управлением привилегированного пользователя или как системные сервисы.

В данном разделе описывается, каким образом можно зарегистрировать в среде Windows NT привилегированного пользователя с правами администратора.

Новый пользователь или, в терминологии NT, пользовательский идентификатор (user ID) создается путем вызова на выполнение программы User Manager.

1. Запустите программу, нажав кнопку **Start** на панели задач, выбрав в главном меню пункт **Programs**, а далее последовательно команды **Administrative Tools (Common)** и **User Manager**.
2. В окне программы User Manager выберите последовательно **User**, **New User**.

   Появится диалоговое окно.
3. Заполните форму, как показано на рис. 5.29.
   - Пользовательское имя не должно превышать 8-ми символов, поэтому если вы будете проводить инсталляцию под именем Administrator, то далее работа с DB2 окажется невозможной.
   - Для тестирования обычно используется имя db2admin.
   - Пользовательский идентификатор не должен превышать 8-ми символов.
   - Отметьте флажок **Password Never Expires**, чтобы система не заставляла каждый раз изменять его.
4. Щелкните кнопку **Groups**, чтобы добавить нового пользователя в группу администраторов (рис. 5.30).

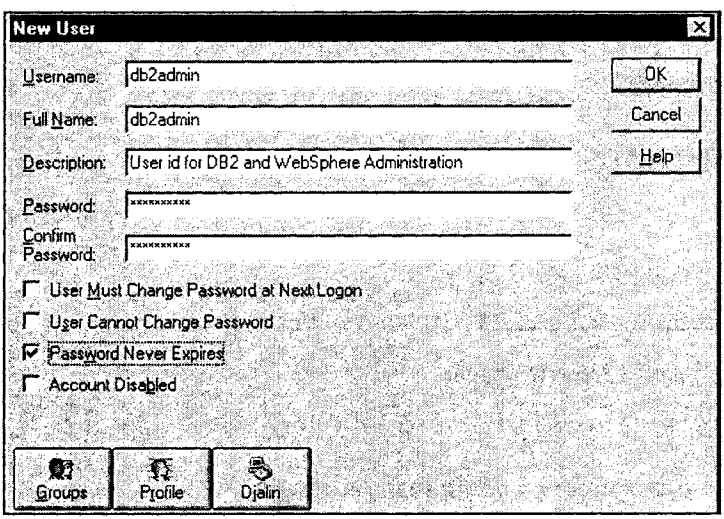


Рис. 5.29. Добавление нового пользователя для выполнения DB2, WebSphere и Domino

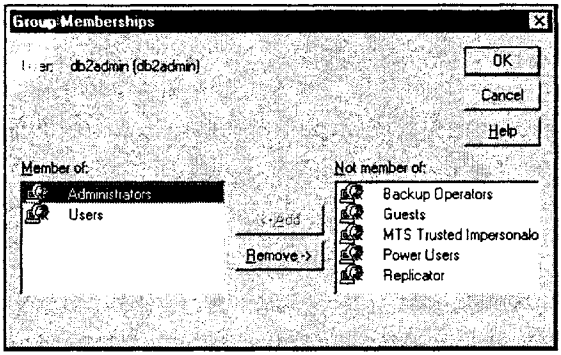


Рис. 5.30. Добавление нового пользователя в группу Administrators

5. Выберите последовательно команды **Policies**, **User Rights** из меню NT User Manager, чтобы назначить только что зарегистрированному пользователю права.

   Нажмите кнопку **Add**. Появится диалоговое окно **Add Users and Groups**.

   Щелкните по кнопке **Show Users**. Выберите пользовательский идентификатор db2admin в списке и нажмите кнопку **Add**. Вернитесь в окно диалога нажатием кнопки **OK**.

   Убедитесь, что наш пользователь (db2admin) внесен в список **Grant To**, отметьте флажок **Show Advanced User Rights** и затем выберите вариант **Act aspart of the operating system** в раскрывающемся списке, как показано на рис. 5.31.

   Нажмите кнопку **OK**, чтобы подтвердить сделанные изменения.

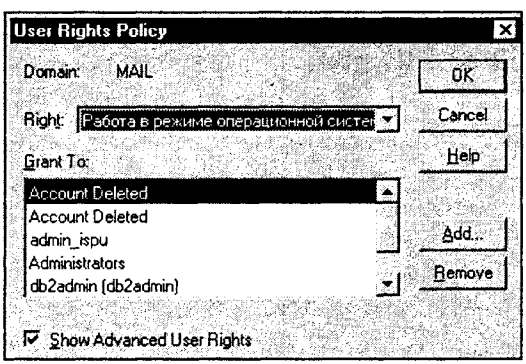


Рис. 5.31. Установка дополнительных прав

6. Закройте окно программы User Manager.
7. Перезапустите Windows NT перед тем, как вы зарегистрируетесь в качестве нового пользователя.

После этого следует по отдельности проинсталлировать:

- DB2 UDB Release 6.1
- IBM JDK 1.1.7P

- Domino Server R5
- WebSphere V3.02

и позаботиться о том, что бы все необходимые архивы и библиотеки были видны в системе.

Приложения Domino и WebSphere могут эксплуатироваться вместе, даже если в сети используется еще один HTTP-сервер, например Microsoft IIS. WebSphere поддерживает несколько различных HTTP-серверов. Один из них — Microsoft IIS, который также может использоваться в качестве HTTP-сервера для Domino R5. В данный момент Domino поддерживает MS IIS и собственный HTTP-сервер; в будущем он будет поддерживать Apache (и, возможно, Netscape) HTTP-сервер. Сегодня, однако, лучшим вариантом использования HTTP-сервера является тот, при котором поддерживаются оба продукта — и WebSphere и Lotus Domino.

Конфигурирование IIS для Domino

Чтобы сконфигурировать IIS для использования его в качестве HTTP-сервера Domino, выберите **ISAPI filters** в окне управления сервисами IIS и нажмите кнопку **Add**.

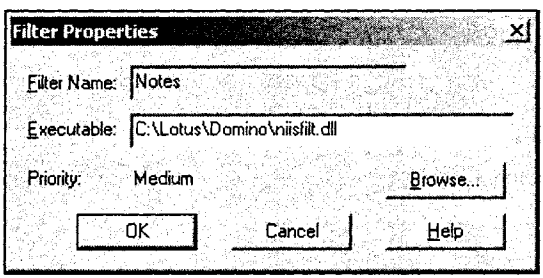


Рис. 5.32. Добавить фильтр Domino к IIS

Вы должны указать в поле **Executable** путь к вашему каталогу, в котором инсталлирован Domino-сервер (рис. 5.32). Вы также можете воспользоваться кнопкой **Browse**, чтобы найти требуемый каталог с файлом niisfilt.dll.

Когда выбранный встраиваемый модуль (plug-in) инсталлирован и загружен, он будет отображаться в списке на конфигурационной вкладке, как показано на рис. 5.33.

Чтобы заставить IIS распознавать файлы с расширением .nsf, которые управляются Domino-сервером, необходимо выбрать **Home Directory** на главной панели определения свойств (Master Properties Panel). Затем надо нажать кнопку **Configuration** и, далее, кнопку **Add** в появившемся диалоговом окне. После этого следует указать требуемое для Domino ISAPI-расширение, как показано на рис. 5.34.

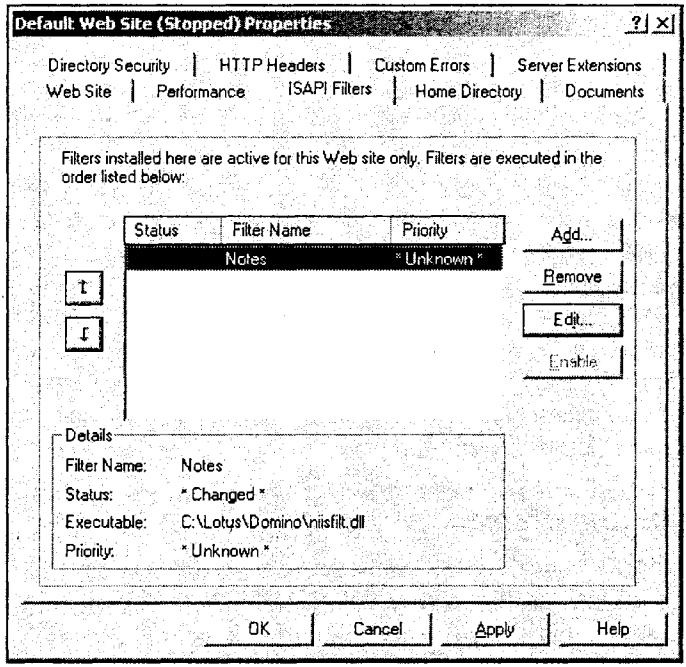


Рис. 5.33. Инсталлированные IIS-фильтры

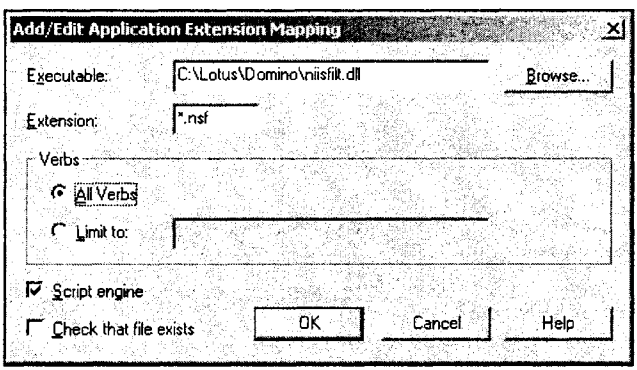


Рис. 5.34. Добавление Domino ISAPI-расширения

Теперь снова надо проверить, правильный ли путь к программам Domino внесен в список, после того как список обрабатываемых расширений будет показан в окне **Application Configuration** (рис. 5.35).

Теперь необходимо возвратиться в Microsoft Management Console и определить новый виртуальный каталог.

1. Щелкните значок "+" слева от имени хоста в окне **Internet Information Server** на левой панели. По щелчку правой кнопкой мыши от **Default Web Site** выберите последовательно команды **New**, **Virtual Directory**.

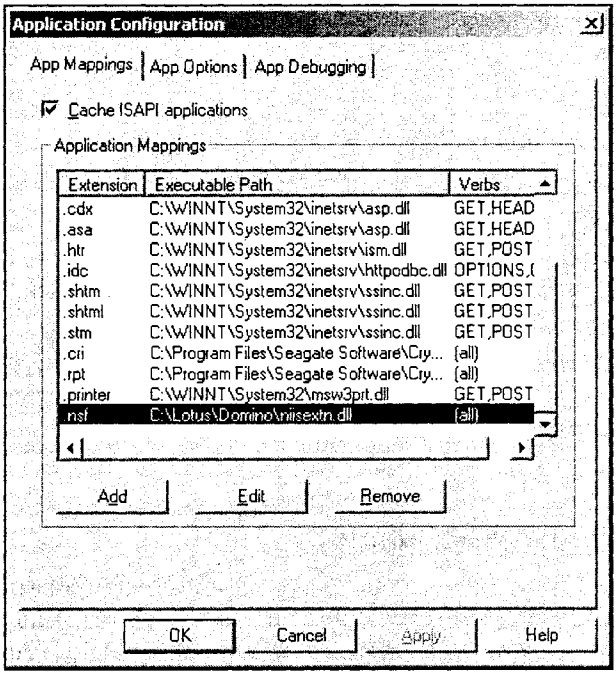


Рис. 5.35. Список ISAPI-расширений и Application Mappings

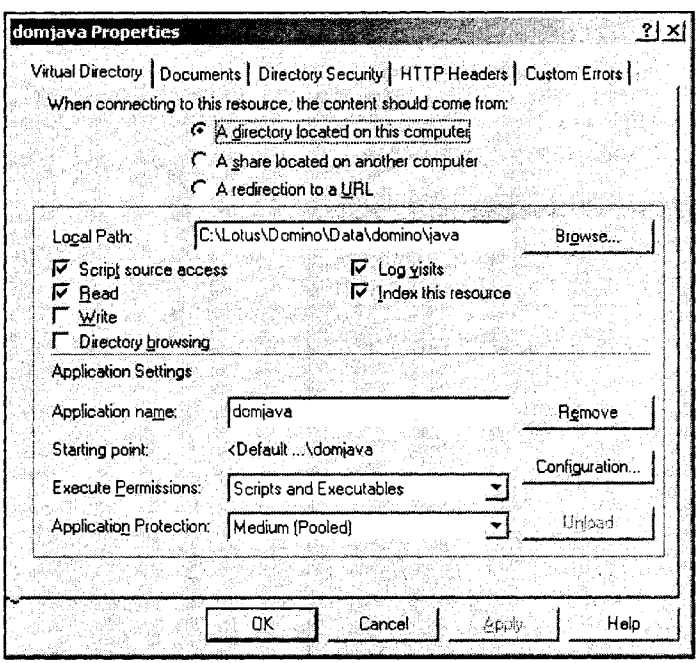


Рис. 5.36. Окно определения виртуального каталога

2. Введите `icons` в поле **Application name** и затем полный путь к необходимому каталогу Domino. В нашем примере это будет

   `C:\Lotus\domino\data\domino\icons`

3. Нажмите кнопку **Next**, а затем кнопку **Finish** на управляющей панели.

Этим способом добавляется каталог Domino в качестве виртуального каталога для IIS, который будет использоваться Domino-приложениями.

Точно таким же способом добавим еще один каталог Domino, повторив вышеописанную процедуру и указав в качестве виртуального каталога domjava, определив путь к Domino как C:\Lotus\Domino\Data\domino\java (рис. 5.36).

Теперь вы должны протестировать возможность совместной работы IIS и Domino. Запустите IIS-сервер и убедитесь, что Domino HTTP-задача выгружена. Из Domino выдается для этого команда:

```
tell http quit
```

Поддержка сервлетов IIS для WebSphere

Чтобы использовать IIS вместе с WebSphere, необходимо проинсталлировать специальный файл с именем PQ37562.zip. Его можно найти по многим адресам в Интернете. Он содержит новую версию библиотеки iis20.dll. Вы должны остановить Domino, WebSphere и IIS и скопировать iis20.dll файл в WebSphere каталог bin. В нашем примере пусть это будет D:\WebSphere \AppServer\bin.

Конфигурирование IIS для WebSphere

Вы можете сконфигурировать IIS для WebSphere при помощи специальной утилиты iis40cfg, входящей в поставку WebSphere.

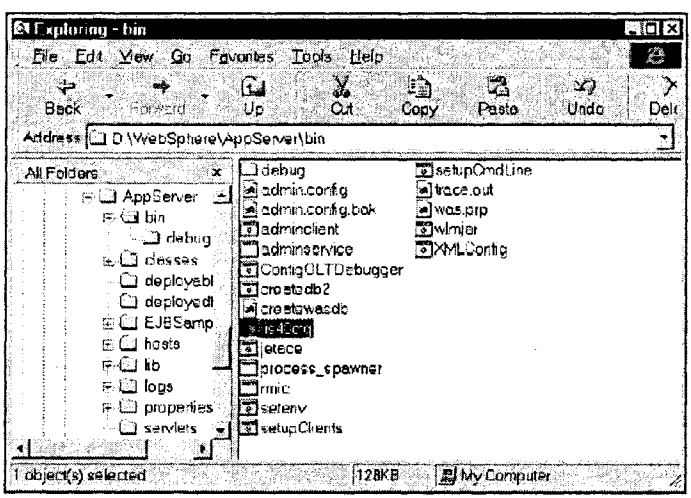


Рис. 5.37. Запуск программы iis40cfg

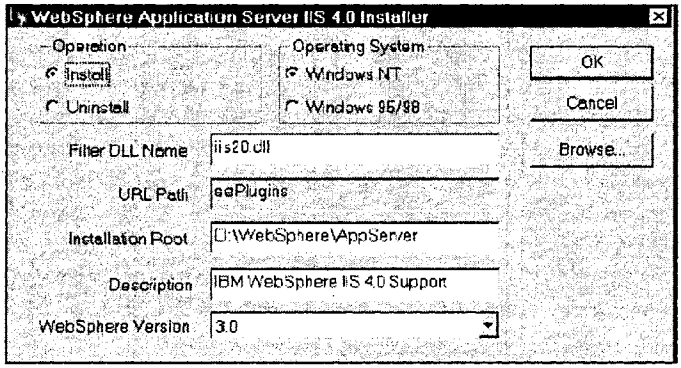


Рис. 5.38. Диалоговое окно **WebSphere Application Server IIS 4.0 Installer**

Запустите Windows NT Explorer, выберите каталог bin в папке WebSphere. По умолчанию это будет каталог \WebSphere\AppServer\bin на том устройстве, где проинсталлирован WebSphere. Выбрав указанную программу, надо просто запустить ее (рис. 5.37).

Сразу после своего запуска программа выдает диалоговое окно, представленное на рис. 5.38.

Если информация, которую вы увидите в данном диалоговом окне, корректна, просто нажмите кнопку **OK**, если нет — укажите правильные параметры.

Отметим, что WebSphere также добавляет виртуальный каталог с именем IBMWebAS с подкаталогом Samples.

Сервер приложений Oracle iAS

Что такое Oracle iAS

Интернет почти сразу вышел за рамки традиционной двухуровневой архитектуры клиент-серверных приложений и стал базироваться на многоуровневой архитектуре разработки серверных приложений, где уровень представлений данных изначально отделен от бизнес-логики. Oracle Internet Application Server обеспечивает идеальную среду для разработки всех типов Интернет-приложений.

Oracle iAS образует набор служб, предлагаемых к использованию для проектирования распределенных, многоуровневых, кластерных и масштабируемых приложений, которые могут функционировать почти на всех аппаратно-программных платформах. Рис. 5.39 иллюстрирует основные элементы Oracle iAS, разбитые на пять основных категорий: коммуникационные сервисы (Communication Services), представления (Presentations), бизнес-логика (Business Logic), управление данными (Data Management) и системные сервисы (System Services). В добавление, Oracle iAS включает набор разрабатывающих средств, не показанных на этом рисунке.

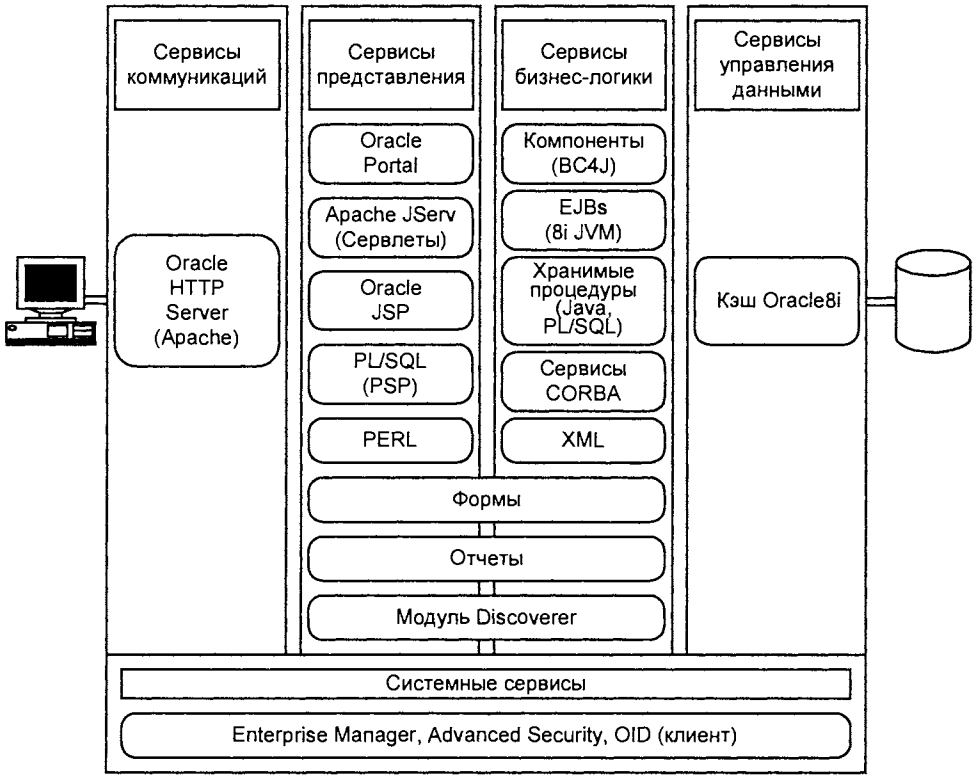


Рис. 5.39. Основные элементы Oracle iAS

Теперь дадим краткое описание каждой службы, входящей в структуру Oracle iAS, чтобы составить общее представление об этой грандиозной архитектуре.

Коммуникационные сервисы

Эти службы отвечают за обработку входящих запросов на соединение, поступающих от удаленных и локальных клиентов. Одни из этих запросов обрабатываются Oracle HTTP Server, а другие передаются к иным частям Oracle Internet Application Server. Эти компоненты Oracle iAS поддерживают разнообразные сетевые протоколы, включая, IIOP, RMI и Net8. В добавление, Oracle iAS может использовать новейшие мобильные WAP-технологии.

Oracle HTTP Server

Oracle HTTP Server основан на Apache-сервере, о котором многое уже говорилось. Здесь стоит отметить, что мировые компьютерные гиганты отнюдь не гнушаются использовать в своих разработках вещи, которые возникли к

жизни в нон-корпоративной среде. Apache, созданный энтузиастами и любителями, относится именно к таким продуктам и ни одна фирма в мире не может изобрести ничего лучшего, чем этот пионер мира Web. Мне иногда становятся смешны снобы, работающие в области высоких (high tech) технологий и делающие гримасу, услышав о выдающихся достижениях одиночек. Именно стараниями таких одиночек, программирующих на C++, а теперь на Java, и создана современная среда Интернета. Именно их усилиями движется вперед прогресс в создании принципиально новых систем программирования, а вовсе не стандартным конвейерным потоком в виде пакетов прикладных программ с "наклейками" нескольких десятков фирм, имена которых у всех на слуху. Общепризнанным является факт, что более 60% всех мировых Интернет-сайтов базируются на Apache. Представленная на рис. 5.40 диаграмма иллюстрирует компоненты, входящие в состав Oracle HTTP Server.

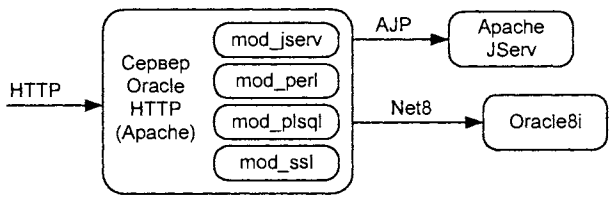


Рис. 5.40. Oracle HTTP Server

Модули Oracle HTTP Server

Mods — это специальные встраиваемые к HTTP-серверу модули, которые расширяют его функциональность путем применения либо стандартных встроенных сервисов типа `mod_ssl`, либо используя внешние процедуры типа `mod_jserv`, которые реализует Java-обработку через Apache JServ. В добавление к компилируемым модулям Apache Oracle HTTP Server имеет несколько своих специфических модулей, которые описываются ниже.

mod_plsql

Этот модуль обслуживает HTTP-запросы для сохраненных процедур в СУБД Oracle8i.

Модуль `mod_plsql` (рис. 5.41) выполняется как отдельный процесс HTTP Server и является версией PL/SQL Cartridge.

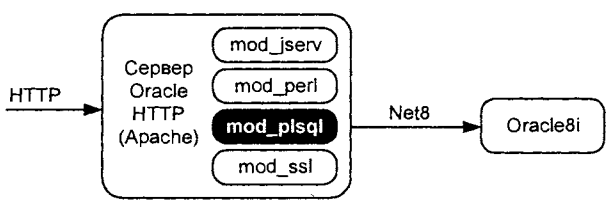


Рис. 5.41. Модуль `mod_plsql`

mod_ssl

Данный модуль (рис. 5.42) обеспечивает стандартный HTTPS-протокол, который полностью поддерживается в Oracle. Он позволяет шифровать сообщения, передаваемые через Интернет посредством механизма Secure Sockets Layer (SSL). Также может использоваться и цифровая подпись.

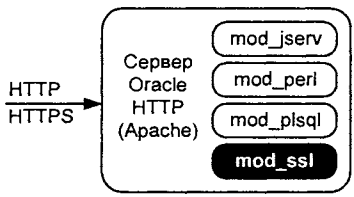


Рис. 5.42. Модуль `mod_ssl`

mod_perl

Этот модуль передает прикладные Perl-запросы к Perl-интерпретатору, который встроен в Oracle, что, опять-таки, свидетельствует об огромном внимании разработчиков Oracle к миру программистов-индивидуалов, в чьей среде необыкновенно популярен Perl. Встроенный Perl-интерпретатор кэширует процессы, за счет чего модули и сценарии загружаются и компилируются только один раз, позволяя серверу управлять загруженной и оттранслированной программой, а следовательно, экономить дорогостоящие ресурсы.

mod_jserv

Модуль `mod_jserv` обслуживает все Java-сервлеты и JSP-запросы, передавая их к исполняющей машине JServ Servlet Engine через Apache JServ-протокол (AJP). В случае JSP, исполняющая OracleJSP-машина управляет процессом интерпретации и выполнения из среды Apache JServ.

Сервисы представлений

Эти службы в Oracle iAS обычно отвечают за графическое представление данных и чаще всего построены в виде форм на HTML. Oracle iAS поддерживает несколько способов генерации графических форм и активно использует скрипты и Java-сервлеты, встроенные в Oracle Portal Services. Рассмотрим это несколько более подробно.

Oracle Portal

Oracle Portal — это программное средство, облегчающее создание порталов. Любая компания, ведущая часть своего бизнеса через Интернет, нуждается в простых и удобных средствах публикации и активного обновления Web-содержания. Именно для того, чтобы обеспечить персонал компании, не

особенно искушенный в программировании, понятными им средствами обновления информации на различных информационных площадках портала компания Oracle снабдила своих разработчиков специальными средствами в виде Portal Developer's Kit API's для интеграции с другими приложениями через универсальные интерфейсы Java и XML (рис. 5.43).

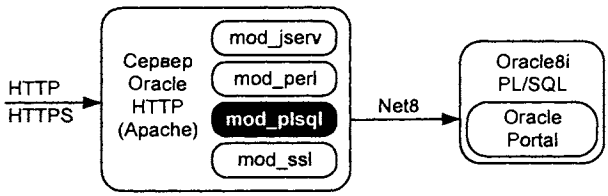


Рис. 5.43. Использование Oracle Portal

Apache JServ

Apache JServ — это исполняющая Java-сервлеты машина, полностью совместимая со спецификацией Sun Microsystems Java Servlet 2.0 API. Подобно Oracle Portal, Apache JServ состоит из двух частей. Первая часть — это исполняющая сервлет-машина, написанная на чистом 100%-Java. Вторая часть — это расширение `mod_jserv`, которое выполняется внутри HTTP-сервера и диспетчеризует запросы HTTP-сервлета к сервлет-машине. При этом используется коммуникационный протокол Apache JServ Protocol (AJP), который работает поверх TCP/IP и позволяет сервлет-машине выполняться либо локально, либо удаленно (рис. 5.44).

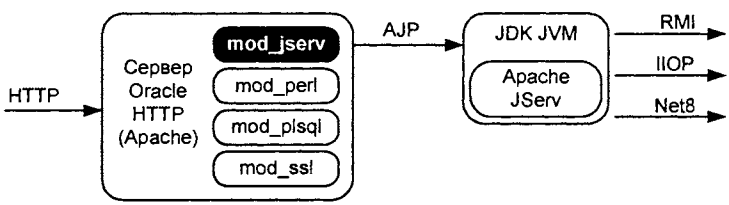


Рис. 5.44. Процесс обработки запроса с использованием Apache JServ

Oracle JSP

Oracle JSP является интерпретатором JSP-страниц. Еще раз подчеркнем, что JSP-страницы используются в основном для разделения наполнения страниц и бизнес-процедур, связанных с этими страницами, позволяя Web-дизайнерам создавать профессионально оформленные страницы, способные динамично изменять свое содержимое, благодаря встроенным в них Java-компонентам, о которых, кроме их названия дизайнер может не иметь никакого представления. Oracle JSP выполняется под управлением Apache JServ или любой другой стандартной сервлет-машины, которая поддержива-

ет сервлет-спецификацию версии 2.0 или выше. Oracle JSP компилирует JSP-страницы в сервлеты и может быть вызван двумя способами: динамически через пользовательский запрос к JSP или непосредственно самим JSP-разработчиком (рис. 5.45). Дополнительно Oracle JSP поддерживает свою собственную схему обработки исключительных событий для СУБД-приложений, множество национальных кодировок (National Language Support, NLS), что немаловажно для тех, кто уже сталкивался с вечными проблемами кириллизации новейших программных технологий, к которым относится JSP.

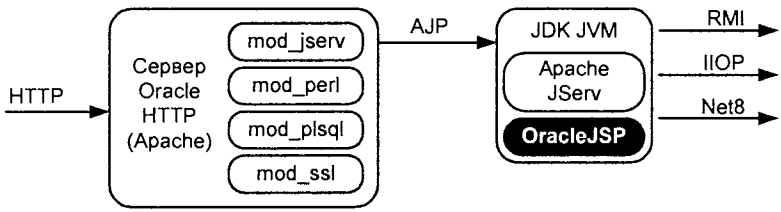


Рис. 5.45. Использование JSP в Oracle

PL/SQL Server Pages

PSP очень похож на Java Server Pages, за исключением того, что в нем вместо Java используется сервер-ориентированный PL/SQL (рис. 5.46). Oracle PSP включает PSP-компилятор и PL/SQL Web Toolkit. PSP может применяться, когда подавляющая часть логики Web-приложения связана с СУБД.

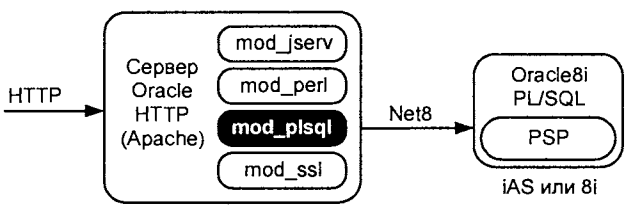


Рис. 5.46. Использование PL/SQL Server Pages

Perl-интерпретатор

Perl — одна из самых излюбленных "игрушек" индивидуумов, двигающих вперед технический прогресс в сфере информационных технологий, не мог не найти место в Oracle, команда разработчиков которого представляет собой наиболее яркое созвездие современных программистов во главе со своим несменяемым лидером — Ларри Эллисоном. Perl непосредственно встроен в Oracle HTTP Server и, таким образом, экономит время, требующееся для запуска внешнего интерпретатора, или порождения лишних IPC-вызовов для выполнения Perl-скриптов. Когда Oracle HTTP Server получает

HTTP-запрос для Perl-скрипта, он передает его в модуль `mod_perl`, который в свою очередь маршрутизирует его к Perl-интерпретатору (рис. 5.47).

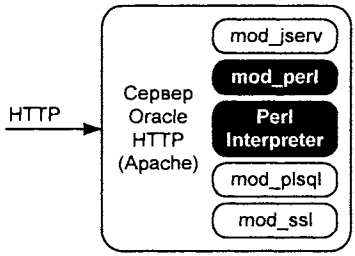


Рис. 5.47. Использование Perl в Oracle

Бизнес-логика

Бизнес-компоненты для Java в Oracle

Oracle Business Components for Java (BC4J) — это 100%-Java и XML-компонентно организованная структура (framework), которая позволяет разрабатывать сложные многоуровневые, распределенные приложения с использованием технологии встраивания повторно используемых компонентов. BC4J можно применять для динамического изменения логики работы приложения без всякой модификации программной части самого приложения. Применение BC4J в Oracle иллюстрирует рис. 5.48.

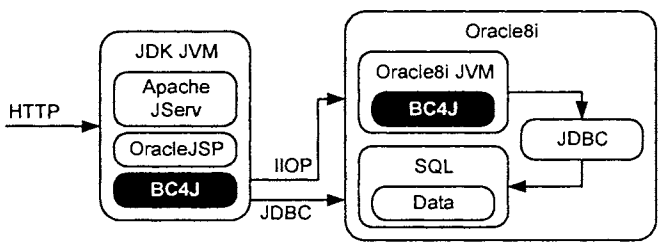


Рис. 5.48. Использование BC4J в Oracle

Поддержка расширенной модели Java

Для разработки высокомасштабируемых, сервер-ориентированных Java-платформ *виртуальная машина Java* в Oracle8i выполнена как 100%-ный Java-совместимый сервер, который поддерживает J2EE (включая сервлеты, JSP и Enterprise JavaBeans), CORBA, а так же сохраненные процедуры для баз данных. Виртуальная машина Oracle8i JVM достигает высокой масштабируемости за счет своей уникальной архитектуры, оптимизации используемой памяти и способности к обработке большого числа конкурентных запросов (рис. 5.49).

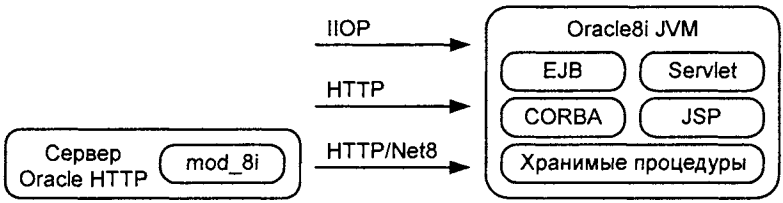


Рис. 5.49. Поддержка J2EE в Oracle

Oracle XML Developer's Kit

Oracle XML Developer's Kit (XDK) содержит библиотеки XML-компонентов и набор утилит, которые могут быть использованы для разработки XML-ориентированных приложений и Web-сайтов. В состав XDK for Java входят XML-парсер, XSL-транслятор, генератор XML-классов, XSQL Servlet, и XML Transviewer Beans. XML-компоненты поставляются в исходных кодах на языках программирования C и PL/SQL.

Программно-аппаратная масштабируемость Oracle

Oracle может функционировать на огромном количестве программно-аппаратных платформ.

Oracle8i поддерживает операционные системы Microsoft, все UNIX-платформы и множество других разнообразных систем.

Пользователи Oracle избавлены от необходимости переписывать свои приложения при миграции с одной платформы на другую. Рис. 5.50 иллюстрирует некоторые возможные платформы для комплексного внедрения решений Oracle.

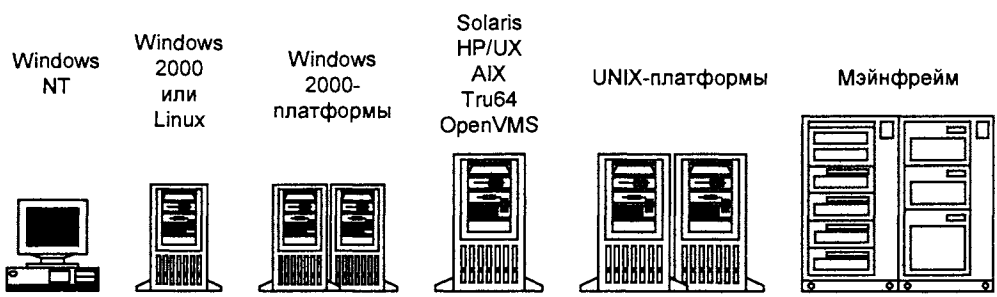


Рис. 5.50. Масштабируемость Oracle от ПК до мэйнфрейма

Протокол SOAP

SOAP (Simple Object Access Protocol) — это открытый протокол, который определяет на основе XML (Extensible Markup Language) общий формат для коммуникационной связи между любыми Интернет-приложениями и сервисами. Его исходная спецификация была разработана прошлой осенью компаниями DevelopMentor Inc., Microsoft и UserLand Software Inc.; в создании версии 1.1 принимали участие корпорации IBM и Lotus Development. Спецификация SOAP предоставляет общий механизм для интеграции сервисов в Интернет и/или интрасетях независимо от применяемой операционной системы, модели объектов или языка программирования. Основанный на Интернет-стандартах XML и HTTP, протокол SOAP позволяет взаимодействовать друг с другом любым новым или существующим приложениям. Web-узлы, поддерживающие SOAP, могут стать Web-сервисами, доступными чисто программным путем и не требующими участия человека.

Приложение к главе 5. Ресурсы и ссылки в Интернете

Ресурсы

Enhydra:

http://www.enhydra.org

http://www.enhydra.com

http://www.lutris.com

Jakarta:

http://jakarta.apache.org/index.html

Apache:

http://www.apache.org

http://apache.lexa.ru

VisualAge for Java:

http://www-4.ibm.com/software/ad/vajava/

IBM VisualAge Developer Domain (VADD):

http://www.software.ibm.com/vadd

IBM WebSphere Application Server:

http://www-4.ibm.com/software/webservers/

The Jakarta Project:

http://jakarta.apache.org/

Sun Microsystems Java Server Pages Technology:

http://www.javasoft.com/products/jsp/index.html

Sun Microsystems Servlet Technology:

http://www.javasoft.com/products/servlet/index.html

Ссылки

Sheldon Wosnick, Debugging Servlets and JSP with WebSphere Application Server and VisualAge for Java.

Sheldon Wosnick, Anita Huang, Developing Server-Side Web Applications with Java Server Pages and VisualAge for Java (Part 1 of 2).

Sheldon Wosnick, Anita Huang, Developing Server-Side Web Applications with Java Server Pages and VisualAge for Java (Part 2 of 2).

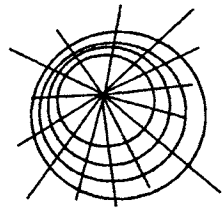
Sheldon Wosnick, VisualAge for Java: New EJB and JSP Tooling.

Sheldon Wosnick, Developing Server-Side Java Servlets with VisualAge for Java.

Joe Winchester, Using the VisualAge for Java Debugger, Part 1.

Joe Winchester, Using the VisualAge for Java Debugger, Part 2.

ГЛАВА 6



Способы связи с базами данных из Интернета и Web-приложений, основанные на традиционных методиках

Концепция ODBC

Зачем нужен ODBC

ODBC-драйверы могут понадобиться в любой операционной системе на сервере или на клиенте, когда вы разрабатываете собственные приложения, которые могут извлекать данные из различных баз данных, хранящихся как на SQL-серверах, так и в специализированных реляционных таблицах. В принципе, таблицы, существующие в виде отдельных файлов (*.dbf (dBase) или *.db (Paradox)), — это уже дела давно минувших дней. Теперь на SQL-сервере можно сохранять все! Однако и отдельные таблицы еще довольно часто используются, чтобы обеспечить доступ к так называемым *legacy systems* — наследуемым системам. Тем не менее в нынешние времена, пожалуй, нет ничего более привлекающего, чем использовать мощный SQL-сервер в качестве Data Warehouse и OLAP-сервера. Приложения для Unix или Windows, написанные на обыкновенном C, Basic или Java могут получать доступ к многочисленным данным в самых различных форматах через ODBC-источники. Эти приложения могут заниматься реконструкцией и преобразованием данных, реализовать различные мосты между данными, Data Pump — т. е. перекачку данных между различными базами и т. д., — словом, обеспечивать складирование и инвентаризацию данных. Поскольку все эти приложения в настоящее время обыкновенно пишутся на языках программирования и с использованием интерфейсов для применения в среде Интернет/интранет, им нет ничего проще, как воспользоваться старой, доброй традицией и для доступа к базам данных применять все те же, проверенные временем ODBC-драйверы. Освежим в памяти концепцию ODBC.

На рис. 6.1 схематически изображен эволюционный процесс разработки SQL-ориентированных приложений.

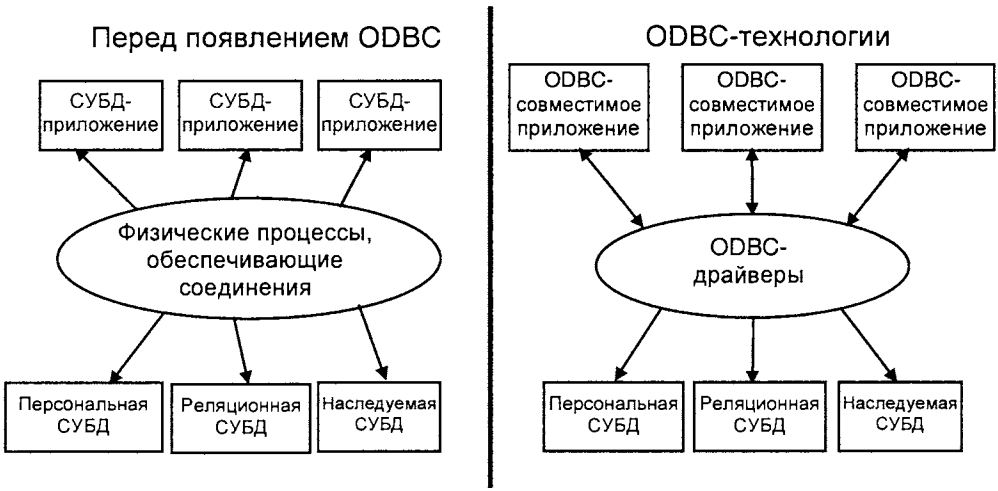


Рис. 6.1. Эволюция доступа к базам данных

Приведенный рисунок иллюстрирует, как изменился подход. Вместо разработки в каждом отдельном случае уникального интерфейса доступа к различным базам данных теперь можно воспользоваться одним общим интерфейсом, позволяющим взамен трех сложных СУБД-приложений написать одно, и достаточно простое.

ODBC Driver Manager

ODBC обеспечивает общий API-интерфейс для доступа к самым разнообразным базам данных. Этот интерфейс был разработан совместно двумя комитетами X/OPEN и SQL Access Group(SAG) и получил первоначальное название как стандарт CLI (Call Level Interface).

Написать приложение, использующее стандартный интерфейс для доступа к любой произвольной базе данных всегда и проще и экономнее, чем применять конкретные, так называемые "родные" интерфейсы, которые быстро устаревают.

В концепции ODBC всегда используется один ODBC Driver Manager и несколько ODBC-драйверов или разделяемых библиотек для доступа к конкретным базам данных (рис. 6.2).

ODBC Driver Manager — является центральным компонентом, который связывает ODBC-совместимые приложения и разделяемые объекты (вспом-

ните, что в MS Windows — это обычная библиотека DLL), которые выполняют специфические функции по обработке SQL-запросов к конкретной БД.

Главное при этом заключается в том, что подобный подход является вполне универсальным и механизм ODBC может работать практически в любой операционной системе.

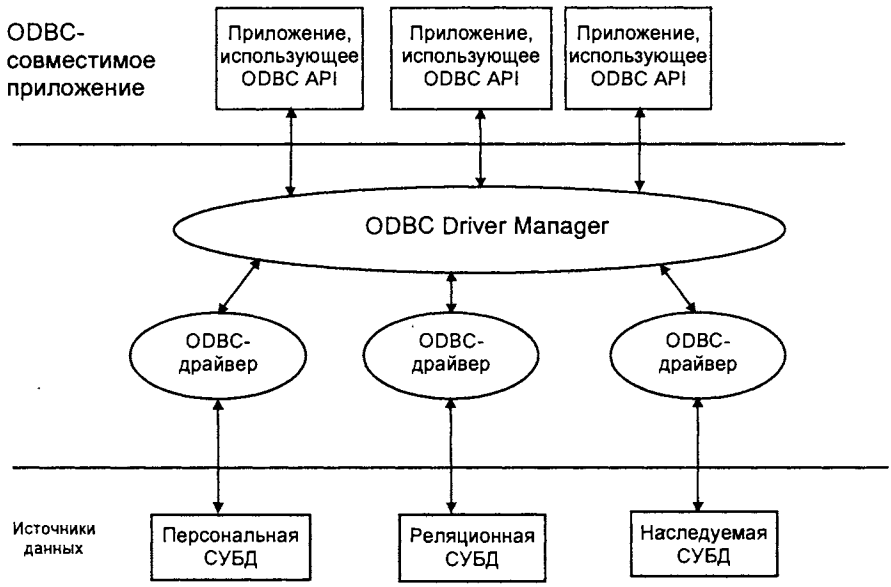


Рис. 6.2. Структура ODBC

Построение стандартных ODBC-приложений в Microsoft Windows 95/98/NT

Главными компонентами ODBC в MS Windows являются:

- ODBC.INI;
- ODBC.DLL;
- индивидуальные ODBC-драйверы для каждой базы данных.

Файл ODBC.INI обеспечивает информацией о базах данных, к которым вы обращаетесь через ODBC. В первой секции INI-файла указывается список всех баз и их типов, например:

```
[ODBC 32 bit Data Sources]
MS Access 97 Database=Microsoft Access Driver (*.mdb) (32 bit)
Excel Files=Microsoft Excel Driver (*.xls) (32 bit)
FoxPro Files=Microsoft FoxPro Driver (*.dbf) (32 bit)
```

```
Paradox Files=Microsoft Paradox Driver (*.db) (32 bit)
Text Files=Microsoft Text Driver (*.txt; *.csv) (32 bit)
mage=Microsoft Access Driver (*.mdb) (32 bit)
Reestr_Sybase=Sybase SQL Anywhere 5.0 (32 bit)
dBASEFile=INTERSOLV 3.01 32-BIT dBASEFile (*.dbf) (32 bit)
Sybase_demo=Sybase SQL Anywhere 5.0 (32 bit)
Dynamo Demo=Sybase SQL Anywhere 5.0 (32 bit)
dBase1=Microsoft dBase Driver (*.dbf) (32 bit)
dBase=Microsoft dBase Driver (*.dbf) (32 bit)
LNotes=Lotus NotesSQL 2.0 (32-bit) ODBC Driver (*.nsf) (32 bit)
My_InterBase=InterBase 4.x Driver by Visigenic (*.gdb) (32 bit)
OrderEntryDB=Microsoft Access Driver (*.mdb) (32 bit)
```

Каждый источник данных списка затем описывается в указанной секции более подробно:

```
[MS Access 97 Database]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[Excel Files]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[FoxPro Files]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[Paradox Files]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[Text Files]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[mage]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[ODBC Data Sources]
SQL Anywhere 5.0 Sample Client=Sybase SQL Anywhere 5.0

[SQL Anywhere 5.0 Sample Client]
driver=d:\sqlany50\win\wod50w.dll
description=Sybase SQL Anywhere Client/Server
EngineName=place_server_name_here
Start=d:\sqlany50\win\dbclienw

[Reestr_Sybase]
Driver32=d:\SQLAny50\win32\wod50t.dll

[dBASEFile]
Driver32=C:\WINDOWS\SYSTEM\IVDBF12.DLL
```

```
[Sybase_demo]
Driver32=C:\SQLAny50\win32\wod50t.dll

[Dynamo Demo]
Driver32=d:\SQLAny50\win32\wod50t.dll

[dBase1]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[dBase]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[LNotes]
Driver32=C:\WINDOWS\SYSTEM\nsql32.dll

[My_InterBase]
Driver32=C:\WINDOWS\SYSTEM\iscdrv32.dll

[OrderEntryDB]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll
```

Следует сказать, что файл ODBC.INI для Windows давно уже формируется не вручную, как раньше, а с помощью интерактивной программы ODBCAD32.EXE (так называемый ODBC-администратор).

К примеру, MS Access 97 Database описывается соответствующим ему драйвером в секции с тем же именем, как:

```
[MS Access 97 Database]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll
```

В прежние времена в файл ODBC.INI заносилась и другая критически важная информация, например местоположение базы данных для данного ODBC-источника, используемая машина базы данных, такая как MS Jet и др., например:

```
DefaultDir=C:\SUPERAPP
JetIniPath=MSACC20.INI
DBQ=SUPERAPP.MDB
UID=Admin
Description= SuperApp Database
DataDirectory=c:\SUPERAPP\SUPERAPP.mdb
SingleUser=False
UseSystemDB=True
FileType=RedISAM
SystemDB=c:\SUPERAPP\system.mda
```

Но теперь эта информация сохраняется в системном реестре Windows:

```
HKEY_CURRENT_USER\SOFTWARE\ODBC\ODBC.INI
```

Для того чтобы использовать базу данных с ODBC в программе на С, вы должны выполнить следующее:

1. Инициализировать среду ODBC.

   Функция SQLAllocEnv выделяет для этого память.

   ```
   return (SQLAllocEnv(&hEnv) == SQL_SUCCESS);
   ```

2. Присоединиться к базе данных.

   Функция SQLAllocConnect выполняет присоединение или возвращает ошибку в том случае, если соответствующий драйвер не найден или некорректен.

   ```
   if ((nResult = SQLAllocConnect(hEnv, (HDBC FAR *)&hDBC))
       != SQL_SUCCESS) {
       ... сообщение об ошибке ...
   }
   nResult = SQLConnect(hDBC, (LPSTR)szDBName,SQL_NTS,
       (LPSTR)szUserName,SQL_NTS,
       (LPSTR)szPassword,SQL_NTS);
   if (nResult != SQL_SUCCESS && nResult !=
       SQL_SUCCESS_WITH_INFO) {
       ... сообщение об ошибке...
   }
   ```

3. Выполнить запрос к базе данных.

   Запрос использует SQL-операторы. К примеру, чтобы обновить БД вы вначале выполняете функцию allocation для того, чтобы получить идентификатор для выполнения SQL-оператора:

   ```
   if ((nResult = SQLAllocStmt(hDBC,&hStatement))
       != SQL_SUCCESS) {
       ... сообщение об ошибке...
   }
   ```

   Затем вы должны построить SQL-оператор, который надо выполнить. Например, следующий оператор изменяет таблицу с именем MACHINE, корректируя значение полей (столбцов в SQL) HardwareVersion, SoftwareVersion записи со значением поля MachineID равным значению поля szMachineID:

   ```
   // ODBC SQL statement handle
   HSTMT hStatement = SQL_NULL_HSTMT;
   sprintf(pUpdate, "UPDATE DISTINCTROW MACHINE SET "
           " MACHINE.HardwareVersion = %d,"
           " MACHINE.SoftwareVersion = %d"
           " WHERE (\"MachineID\"=%s);",
           HardwareVersion, SoftwareVersion, szMachineID);
   ```

4. Выполнить SQL-оператор.

```
nResult = SQLExecDirect(hStatement, pUpdate, QL_NTS);
if (nResult != SQL_SUCCESS)
    ... сообщение об ошибке...
```

Следующий фрагмент кода возвращает некоторое количество записей, удовлетворяющих критерию, заданному в таблице EVENT. Поскольку EventID является единственным полем в списке SELECT, только EventID будет доступен для последующей обработки:

```
// ODBC SQL statement handle
HSTMT hStatement = SQL_NULL_HSTMT;
RETCODE nResult;
long lEventID;
sprintf(pSelect, "SELECT DISTINCTION EventID "
        " FROM EVENT WHERE ((MachineID = %s) AND "
        " (Status = 0) AND (EventCode = %d));",
        pQTask->szMachineID, iNoPeriodicalReport);
        nResult = SQLExecDirect(hStatement,pSelect, SQL_NTS);
if (nResult != SQL_SUCCESS) {
    DisplayError(nResult,hStatement);
    SQLFreeStmt(hStatement,SQL_DROP);
    return;
}
```

Нижеприведенный фрагмент связывает поле EventID (поле 0 в списке SELECT) с переменной lEventID:

```
nResult = SQLBindCol(hStatement,1,SQL_C_LONG,
                     &(lEventID), 0,NULL);
if (nResult != SQL_SUCCESS) {
DisplayError(nResult,hStatement);
SQLFreeStmt(hStatement,SQL_DROP);
return;
}
```

Следующий оператор while (в C) выполняется до тех пор, пока не будут выбраны все строки. Каждый раз, когда SQLFetch успешно срабатывает, переменная lEventID получает новое значение.

```
while (TRUE) {
    nResult = SQLFetch(hStatement);
    if (nResult != SQL_SUCCESS) { // no more rows break;
break;
    } {
    ... операции над lEventID ...
}
```

5. Освободить идентификатор-обработчик для SQL-оператора.

   Когда вы закончили выполнение запросов, вы должны освободить идентификатор-обработчик:

   ```
   SQLFreeStmt(hStatement,SQL_DROP);
   ```

6. Прервать соединение с базой данных.

   ```
   SQLDisconnect(hDBC);
   ```

7. Освободить идентификатор-обработчик, выделенный в функции `SQLAllocConnect`.

   ```
   SQLFreeConnect(hDBC);
   ```

8. Освободить среду окружения, распределенную в функции `SQLAllocEnv`.

   ```
   SQLFreeEnv(hEnv);
   ```

Все вышеописанные действия с исчерпывающей подробностью изложены в документах комплекта инструментов разработчика ODBC SDK, которое совершенно свободно распространяется со многих узлов Интернета и доступно по сети Microsoft Developer's Network (MSDN).

Различные стандарты ODBC

Стандарт ODBC 2.0, затем ODBC 2.5 и ODBC 3.x (OLE DB — это новое революционное развитие стандарта), был впервые реализован фирмой Microsoft и получил во всем мире самое широкое распространение. Сама Microsoft выпустила комплект для разработчиков под названием ODBC SDK, где приведено достаточно много примеров программирования с использованием ODBC. Там находится достаточно интересный Help-файл, который дает полное представление обо всех механизмах ODBC. Наиболее сложными с точки зрения программиста являются случаи, когда прикладная программа хочет работать сразу с несколькими драйверами, чтобы получать доступ и обмениваться данными сразу с несколькими базами данных. Такие программы могут понадобиться при программировании различных систем сбора информации из различных источников, агрегирования данных и записи их в интегрированную базу данных.

ODBC-решения

Когда прикладная программа нуждается в специфическом драйвере, она сначала запрашивает список доступных драйверов, чтобы идентифицировать драйвер, и затем требует у Driver Manager загрузить этот драйвер. Driver Manager загружает драйвер и сохраняет адрес каждой функции в драйвере. Чтобы вызвать соответствующую функцию ODBC в драйвере, прикладная программа передает данный запрос Driver Manager и тот вызывает эту функцию по списку сохраненных адресов.

ODBC присуще значительное число особенностей системы управления базами данных, но он не требует, чтобы драйверы поддерживали все из них. Если бы ODBC включал в себя только особенности, свойственные всем без исключения базам данных, то это имело бы ограниченное использование; в конце концов, причина, почему сегодня существует так много различных систем управления базами данных, заключается в том, что все они имеют различные особенности. Тем не менее, ни одна библиотека и ни один универсальный драйвер не может включать в себя все мыслимые возможности.

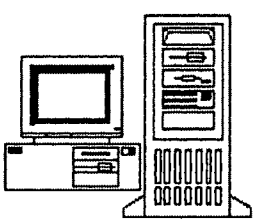
Стандарт ODBC включает значительное число особенностей, гораздо больше чем те, что поддерживаются большинством СУБД, но требует, чтобы драйверы выполняли только оставшееся неохваченным их подмножество. Драйверы делают это в том случае, если данные особенности являются неотъемлемой частью основной СУБД.

Чтобы прикладная программа могла определять то, какие особенности драйвер и СУБД обеспечивают, в ODBC предусмотрены две функции (`SQLGetInfo` и `SQLGetFunctions`), которые возвращают общую информацию относительно драйвера и возможностей системы управления базами данных и списка функций, которые драйвер поддерживает. ODBC также определяет API- и SQL-уровни соответствия грамматики, которые определяют широкие диапазоны особенностей, поддерживаемых каждым конкретным драйвером.

Этим достигается одна важная цель — при модификации и усовершенствовании СУБД меняется только драйвер связи, но не сам ODBC-совместимый интерфейс прикладной программы.

Архитектура ODBC

Существуют различные способы использования ODBC-архитектуры в сетях. Например на рис. 6.3 показано, что все программные компоненты ODBC располагаются на одной машине. Там же находится и база данных.



Система А
Приложение
Модуль Driver Manager
Драйвер (включая ПО для доступа к данным)
Хранилище данных

Рис. 6.3. Конфигурация одной машины

Во втором случае база данных отделена от клиентского компьютера и для связи используется сетевой протокол (обычно TCP/IP или любой другой, который можно выбрать при инсталляции драйвера (рис. 6.4)).

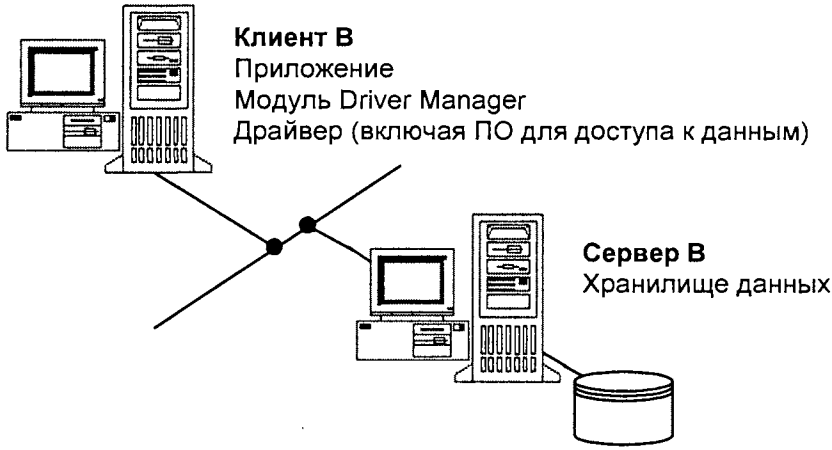


Рис. 6.4. Конфигурация двух машин

Наконец, рис. 6.5 показывает использование еще одного промежуточного слоя `GateWay`, который выполняет чисто служебную задачу и связывается с базой данных по асинхронной линии.

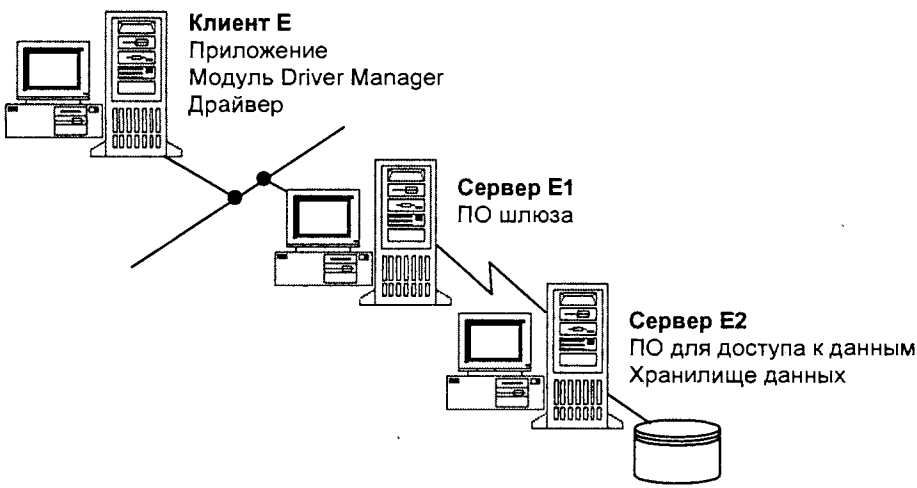


Рис. 6.5. Трехуровневая схема доступа через ODBC

Функциональные возможности стандарта ODBC 3.x

Стандарт Microsoft ODBC 3.x полностью реализует концепцию ODBC ANSI/ISO SQL-92 интерфейса CLI — (Call-Level-Interface) и добавляет многочисленные расширения.

Открытый стандарт доступа к базам данных фирмы Microsoft (ODBC) — это есть, по сути, полная реализация международного стандарта интерфейса CLI для программирования SQL-приложений. ODBC 3.x стал фактическим стандартом программирования SQL-интерфейса. Microsoft использовала международный стандарт SQL Access Group (SAG) как базовый для создания API базы данных Windows. Первоначально SAG развивался как совокупность стандартов X/Open и международных организаций по стандартизации (ANSI и ISO, соответственно). Версия ODBC 3.x, однако, в реализации Microsoft добавляет 300 страниц к основному Справочнику программиста ODBC, что подтверждает огромное увеличение функциональных возможностей.

ODBC 3.x обеспечивает:

- средствами поддержки Unicode (международный стандарт двухбайтового кодирования любых алфавитов);
- улучшенными средствами описания данных;
- средствами обработки ошибок;
- улучшенными средствами обработки больших двоичных и текстовых объектов.

Продукт Microsoft также добавляет пакетные расширения, SQL-92 расширения и приблизительно 20 новых вызовов функций.

Другая впечатляющая новая возможность — поддержка SQL3 локаторов — 32-битовых переменных, получающих значения во время выполнения. Через них обеспечивается доступ к большим двоичным и символьным объектам. Приложения ODBC 2.0 обычно использовали функции `SQLGetData` и `SQLPutData`, чтобы сохранять и читать данные переменной длины. ODBC 3.x включает пять новых функций, которые работают с локаторами и поддерживают функции модификации данных. Приложение получает локатор, вызывая функцию `SQLBindCol` или `SQLGetData`, и находит длину большого объекта, вызывая `SQLGetLength`. ODBC 3.x обеспечивает две функции (`SQLGetPosition` и `SQLGetSubString`) для размещения и извлечения подстроки больших объектных столбцов, содержащих текст. Новая функция `SQLLocator` разрешает приложению захватывать или освобождать локатор после того, как родительская транзакция завершается.

ODBC 3.x содержит и другие возможности, которые не противоречат стандарту SQL3, постепенно заменяющему SQL-92.

ODBC 3.x поддерживает концепцию адаптивного программирования, что означает — можно получить информацию относительно драйвера, базы данных, типов и других возможностей используемого SQL-сервера. Функция `SQLGetTypeInfo` добавляет новые типы данных, в то время как `SQLGetFunctions` возвращает функции, которые этот драйвер поддерживают. ODBC 3.x добав-

ляет почти 40 новых функций, которые являются полезными для записи адаптивного кода.

ODBC 3.x включает ряд функций, чтобы получать специфические метаданные, и добавляет расширенные возможности описания данных. Можно получать информацию каталога относительно таблиц и других объектов, вызывать из прикладной программы функции типа `SQLTables`, `SQLtable`-привилегий, `SQLProcedures`, `SQLprimary`-клавиш, `SQLForeignKeys`, и `SQLStatistics`. Аналогично, приложение может вызывать функции `SQLColumns`, `SQLColAttributes`, `SQLDescribeCol`, `SQLColumnPrivileges` и `SQLProcedureColumns` для получения информации об имени, типе и размерности столбца. Описатели ODBC 3.x основаны на SQL-92 и SQL SQL Descriptor Area (SQLDA). Программисты, использующие динамический SQL, должны быть знакомы с процессом распределения дескрипторной области (SQLDA) и использования утверждения DESCRIBE SQL для ее заполнения. Поскольку SQL-92 стандартизировал формат SQLDA, это позволяет ODBC 3.x использовать аналог динамическому SQL. Существуют и другие многочисленные расширения стандарта, на которых здесь не имеет смысла подробно останавливаться. Архитектура ODBC изложена столь подробно, чтобы читателю было ясно, каким образом можно использовать новые подходы, реализованные в интерфейсах Microsoft, которые помогут при разработке Web-приложений на основе архитектуры OLE DB как дальнейшего развития концепции ODBC.

Цель OLE DB состоит в том, чтобы дать возможность приложениям использовать универсальный доступ к данным, сохраняемым с помощью СУБД, и к данным других источников. OLE DB позволяет прикладным программам пользоваться преимуществами технологии баз данных без необходимости передавать сами данные из места их происхождения в СУБД.

OLE DB — это ключевая технология Microsoft, облегчающая доступ к любым данным, в каком бы месте и в каких форматах они бы не находились. Использование открытых стандартов свидетельствует о том, что эта технология может получить большое значение в будущем и не только на тех платформах, на которых традиционно сильна Microsoft. *Подробности программирования Web-приложений с использованием интерфейсов OLE DB — ADO изложены в главе 4 настоящей книги, где описано программирование страниц ASP с помощью доступа к базам данных, и более детально рассматриваются в книге автора "Доступ к базам данных и техника работы в сети" — М.: Диалог-МИФИ, 2000.* Что же касается ODBC, то эта архитектура не умерла, а, наоборот, получила свое неожиданное продолжение для Java-архитектур в их связи с реляционными серверами и базами данных.

Доступ к базам данных из Web-ориентированных Java-апплетов и Java-программ

Java-программы и апплеты с интерфейсом JDBC-ODBC

JDBC (Java Database Connectivity) является не протоколом, а интерфейсом и основан на спецификациях SAG CLI (SQL Access Group Call Level Interface — интерфейс уровня вызова группы доступа SQL).

Сам по себе JDBC работать не может и использует основные абстракции и методы ODBC. Хотя в стандарте JDBC API и предусмотрена возможность работы не только через ODBC, но через прямые связи к базам данных по двух- или трехзвенной схеме (рис. 6.6), эту схему используют гораздо реже, чем повсеместно используемый JDBC-ODBC-Bridge, занимающий центральное место в общей структуре взаимодействия интерфейсов (рис. 6.7).

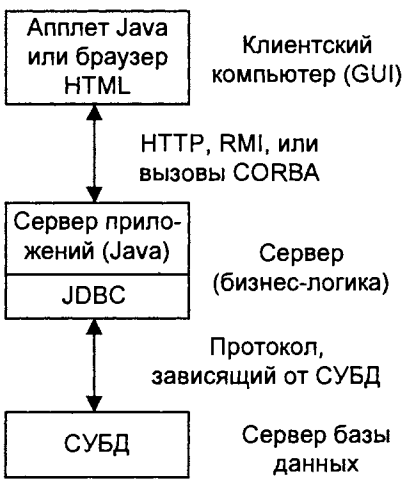


Рис. 6.6. Непосредственный доступ к базе данных по трехзвенной схеме

Даже беглого взгляда на рис. 6.7 вполне достаточно, чтобы понять — общая схема взаимодействия интерфейсов в Java удивительным образом напоминает столь всем знакомую схему ODBC с ее гениальным изобретением драйвер-менеджера к различным СУБД и единого универсального пользовательского интерфейса. JDBC Driver Manager — это основной ствол JDBC-архитектуры. Его первичные функции очень просты — соединить Java-программу и соответствующий JDBC-драйвер и затем выйти из игры. Естественно, что ODBC был взят в качестве основы JDBC из-за его популярности среди независимых поставщиков программного обеспечения и пользователей. Но тогда

возникает законный вопрос — а зачем вообще нужен JDBC и не легче ли было организовать интерфейсный доступ к ODBC-драйверам непосредственно из Java? Ответом на этот вопрос может быть только однозначное *нет*. Путь через JDBC-ODBC-Bridge, как ни странно, может оказаться гораздо короче.

- ODBC нельзя использовать непосредственно из Java, поскольку он основан на C-интерфейсе. Вызов из Java C-кода нарушает целостную концепцию Java, пробивает брешь в защите и делает программу трудно переносимой.
- Перенос ODBC C-API в Java-API нежелателен. К примеру, Java не имеет указателей, в то время как в ODBC они используются.
- ODBC слишком сложен для понимания. В нем смешаны простые и сложные вещи, причем сложные опции иногда применяются для самых простых запросов.
- Java-API необходим, чтобы добиться абсолютно чистых Java-решений. Когда ODBC используется, то ODBC-драйвер и ODBC менеджер должны быть инсталлированы на каждой клиентской машине. В то же время, JDBC-драйвер написан полностью на Java и может легко переноситься на любые платформы от сетевых компьютеров до мэйнфреймов.

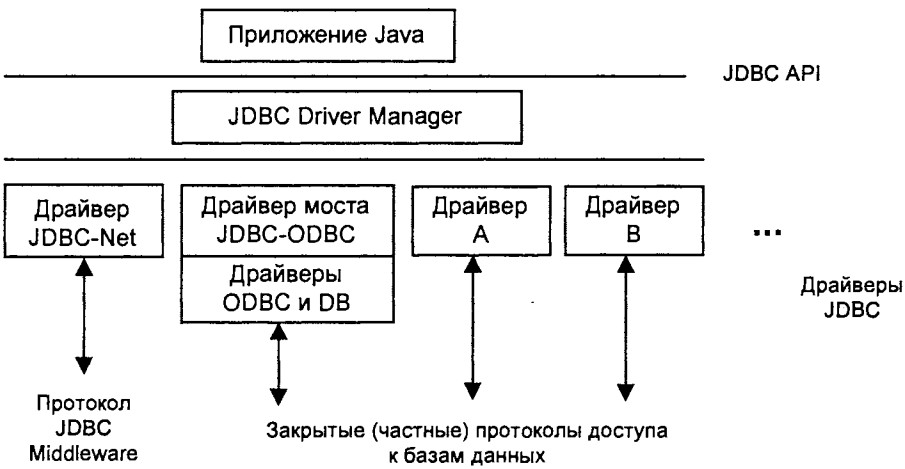


Рис. 6.7. Схема взаимодействия интерфейсов

JDBC API — это естественный Java-интерфейс к базовым SQL-абстракциям и, восприняв дух и основные абстракции концепции ODBC, он реализован, все-таки, как настоящий Java-интерфейс, согласующийся с остальными частями системы Java.

В отличие от интерфейса ODBC, JDBC организован намного проще. Главной его частью является драйвер, поставляемый фирмой JavaSoft для досту-

па из JDBC к источникам данных. Этот драйвер является самым верхним в иерархии классов JDBC и называется *DriverManager*. Согласно установившимся правилам Интернета, база данных и средства ее обслуживания идентифируются при помощи URL.

```
jdbc:<subprotocol>:<subname>
```

где под <subprotocol> понимается имя конкретного драйвера, или некоего механизма установления соединения с базой данных, например, ODBC. В случае применения ODBC, в URL-строку подставляется именно эта аббревиатура, а в качестве <subname> используется обычный DSN (Data Source Name), т. е. имя ODBC-источника из файла ODBC.INI. Например:

```
jdbc:odbc:dBase
```

В некоторых случаях, вместо ODBC может быть использовано имя прямого сетевого сервиса к базе данных, например:

```
jdbc:dcenaming:accounts-payable
```

или

```
jdbc:dbnet://ultra1:1789/state
```

В последнем случае часть URL //ultra1:1789/state представляет собой <subname> и описывает имя хоста, порт и соответствующий идентификатор для доступа к соответствующей базе данных.

Однако, как уже говорилось выше, чаще всего все-таки используется механизм ODBC, благодаря его универсальности и доступности. Программа взаимодействия между драйвером JDBC и ODBC разработана фирмой JavaSoft в сотрудничестве с InterSolv и называется *JDBC-ODBC-Bridge*. Она реализована в виде JdbcOdbc.class (для платформы Windows JdbcOdbc.dll) и входит в поставку JDK1.1. Помимо JdbcOdbc-библиотек должны существовать специальные драйверы (библиотеки) для реализации непосредственного доступа к базам данных через стандартный интерфейс ODBC. Как правило, эти библиотеки описываются в файле ODBC.INI. На внутреннем уровне JDBC-ODBC-Bridge отображает медоды Java в вызовы ODBC и тем самым позволяет использовать любые существующие драйверы ODBC, которых к настоящему времени накоплено в изобилии.

Рассмотрим типичное приложение на Java с доступом к типичному реляционному серверу или даже к обычной dBase-таблице.

```
// Следующий код на Java используется как пример. Простой
// подстановкой соответствующих значений url, login и password, и
// затем подстановкой SQL-операторов вы можете посылать их в базу
// данных.
// ----------------------------------------------------------------
//
```

```
     // Module:      SimpleSelect.java
     //
     // Описание: Эта программа для ODBC API-интерфейса. Java-приложение
     // будет присоединяться к JDBC-драйверу, посылать select-оператор
     // и показывать результаты в таблице
     //
     // Продукт:     JDBC к ODBC Мост
     //
     // Автор:       Karl Moss (модификация для работы с кириллицей
     // С.Дунаева)
     //
     // Дата:        Апрель 1997
     //
     // Copyright:   1990-1996 INTERSOLV, Inc.
     // This software contains confidential and proprietary
     // information of INTERSOLV, Inc.
     //-----------------------------------------------------------------
import java.net.URL;
import java.sql.*;
import java.io.*;

class SimpleSelect {


     public static void main (String args[]) {
             String url   = "jdbc:odbc:dBase";
             String query = "SELECT * FROM my_table";

             try {

                       // Загрузка драйвера jdbc-odbc-bridge

                     Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");

                     DriverManager.setLogStream(System.out);

              // Попытка соединения с драйвером.  Каждый из
              // зарегистрированных драйверов будет загружаться, пока
              // не будет найден тот, который сможет обработать этот URL

                     Connection con = DriverManager.getConnection (
                             url, "", "");

       // Если не можете соединиться, то произойдет exception
      // (исключительная ситуация). Однако если вы попадете
     // в следующую строку программы, значит вы успешно соединились с URL
    // Проверки и печать сообщения об успешном соединении

                     checkForWarning (con.getWarnings ());
```

```
            // Получить DatabaseMetaData-объект и показать
            // информацию о соединении

            DatabaseMetaData dma = con.getMetaData ();

            //System.out.println("\nConnected to " +
                     //dma.getURL());
            //System.out.println("Driver       " +
                    //dma.getDriverName());
            //System.out.println("Version      " +
                    //dma.getDriverVersion());
            //System.out.println("");

            // Создать оператор-объект для посылки
            // SQL-операторов в драйвер

            Statement stmt = con.createStatement ();

        // Образовать запрос, путем создания ResultSet-объекта

            ResultSet rs = stmt.executeQuery (query);

            // Показать все колонки и ряды из набора результатов

            dispResultSet (rs);

            // Закрыть результирующий набор

            rs.close();

            // Закрыть оператор

            stmt.close();

            // Закрыть соединение

            con.close();
        }
        catch (SQLException ex) {

            // Случилось SQLException.  Перехватим и
            // покажем информацию об ошибке.  Заметим, что это
            // может быть множество ошибок, связанных вместе
            //

        //System.out.println ("\n*** SQLException caught ***\n");

        while (ex != null) {
              //System.out.println ("SQLState: " +
                             //  ex.getSQLState ());
             //System.out.println ("Message:  " + ex.getMessage ());
```

```
                //System.out.println ("Vendor:    " +
                                    //ex.getErrorCode ());
                    ex = ex.getNextException ();
                    //System.out.println ("");
                    }
            }
            catch (java.lang.Exception ex) {

             // Получив некоторые другие типы exception, распечатаем их

                    ex.printStackTrace ();
            }
    }

   //-----------------------------------------------------------------
   // checkForWarning
   // Проверка и распечатка предупреждений.  Возврат значения true, если
   // предупреждение существует
   //-----------------------------------------------------------------

    private static boolean checkForWarning (SQLWarning warn)
                        throws SQLException  {
            boolean rc = false;

            // Если SQLWarning-объект был получен, показать
            // предупреждающее сообщение.

            if (warn != null) {
                   System.out.println ("\n *** Warning ***\n");
                    rc = true;
                    while (warn != null) {
                            //System.out.println ("SQLState: " +
                                    //warn.getSQLState ());
                            //System.out.println ("Message:  " +
                                    //warn.getMessage ());
                            //System.out.println ("Vendor:   " +
                                    //warn.getErrorCode ());
                            //System.out.println ("");
                            warn = warn.getNextWarning ();
                    }
            }
            return rc;
    }

    //-----------------------------------------------------------------
    // dispResultSet
    // Показать таблицу полученных результатов
    //-----------------------------------------------------------------
```

```
    private static void dispResultSet (ResultSet rs)
            throws SQLException, IOException
    {
            // Объявление необходимых переменных и
           // константы для желаемой таблицы перекодировки данных
            int i, length, j;
            String cp1 = new String("Cp1251");

            // Получить данные ResultSetMetaData.  Они будут
            // использованы для печати заголовков

            ResultSetMetaData rsmd = rs.getMetaData ();

            // Получить номер столбца в результирующем наборе

            int numCols = rsmd.getColumnCount ();

            // Показать заголовок столбца

            for (i=1; i<=numCols; i++) {
                    if (i > 1) System.out.print(",");
                    //System.out.print(rsmd.getColumnLabel(i));
            }
            System.out.println("");

            // Показать данные, загружая их до тех пор, пока не
            // исчерпается результирующий набор

            boolean more = rs.next ();
            while (more) {

                    // Цикл по столбцам

                    for (i=1; i<=numCols; i++) {

// Следующая группа операторов реализует функции перекодировки
// строк из таблицы базы данных в желаемый формат, потому что в
// различных базах символы могут быть закодированы произвольным
// образом. Если использовать стандартный метод — getString  — на выходе
// получается абракадабра. Строки  нужно сначала перевести в Unicode,
// затем конвертировать в строку Windows и убрать лидирующие нули

            InputStream str1 = rs.getUnicodeStream(i);
             byte str2[];
             byte str3[];
            int sizeCol = rsmd.getColumnDisplaySize(i);
            str2 = new byte[sizeCol+sizeCol];
            str3 = new byte[sizeCol+sizeCol];
            length = str1.read(str2);
```

```
            // Здесь нужно убрать нули из строки, которые предваряют каждый
            // перекодированный символ
               k=1;
               for (j=1; j<sizeCol*2; j++) {
                  if (str2[j] != 0)  {
                        str3[k]=str2[j]; k=k+1; } }

              String str = new String(str3,cp1);
              System.out.print(str);
                        }
                        System.out.println("");

                        // Загрузка следующего ряда в наборе

                        more = rs.next ();
               }
      }

}
```

В этой простой программе, приводимой во множестве руководств, мною произведено одно небольшое изменение, позволяющее использовать ее для работы с различными базами данных, содержащими таблицы с полями в кириллической кодировке. Дело в том, что хотя Java автоматически производит преобразования из Unicode и обратно в соответствии с установленными на вашей машине языковыми спецификациями (так называемые locale), эти преобразования не всегда действуют по отношению к кириллическим шрифтам, особенно когда кириллические строки прописаны не непосредственно в Java-программе, а передаются из внешних источников, например из баз данных, через несколько промежуточных слоев. Та же проблема, как мы увидим далее, возникает и при использовании сервлетов, работающих в тесной взаимоувязке с Web-серверами.

Специально ориентированные Java-приложения

RMI-приложения

RMI (Remote Method Invocation), вызов удаленных методов, обеспечивает средства коммуникации между Java-программами, даже если они выполняются на разных компьютерах, находящихся в противоположных точках земного шара.

Важная особенность RMI заключается в том, что он представляет программируемый интерфейс для работы с сетями в отличие от сокетов TCP. Главное преимущество его в том — что он предлагает вам интерфейс более вы-

сокого уровня, основанный на вызовах методов так, как если бы удаленный объект обрабатывался локально. RMI более удобен и более естественен, чем интерфейс, основанный на сокетах, но он требует выполнения Java-программ на обоих концах соединения. Сетевое соединение, тем не менее достигается использованием все того же протокола TCP/IP.

Рассмотрим основные шаги для построения работающего RMI-приложения:

- Разработка удаленных объектов и кодов для сервера и клиента
- Java-компиляция
- RMI-компиляция
- Перемещение class-файлов в соответствующие каталоги
- Регистрация
- Старт сервера
- Старт клиента

Следующий ниже пример разработан для клиента и сервера, выполняющихся на одной и той же машине, однако, заложенные принципы можно с успехом применить для программ, выполняющихся на разных машинах, и даже с различной архитектурой. Программа клиента ищет имена и номера телефонов в центральной базе, постоянно резидентной на сервере.

Как работает RMI

Вы определяете Java-интерфейс, чтобы описать каждый объект, который будет дистанционно разделяем, и перечисляете общие методы, которые могут быть вызваны для объекта. Сервер будет использовать RMI-интерфейс и создаст объекты для вызова, специальным образом зарегистрированные и доступные для вызова по URL-основанной схеме, например:

```
rmi://localhost/LookupServer
```

Клиент, используя эту запись, будет пытаться отыскать объект с данным именем и получить удаленную ссылку к нему. Затем вызванный метод будет обработан с помощью RMI-компилятора и преобразован из пользовательского кода в последовательную форму объекта, который передается пользователю с помощью протокола TCP/IP.

Разработка удаленного объектного кода

RMI поддерживает объекты Java, осуществляющие связь через их методы, независимо от того, где эти объекты размещены. Первый шаг для создания класса, к которому можно обратиться дистанционно, — это определение интерфейса, описывающего дистанционно-разделяемые методы.

```
// Lookup.java

import java.rmi.*;
```

```
public interface Lookup extends Remote {
   public String findInfo(String info) throws RemoteException;
         }
```

Java.rmi.Remote — пустой интерфейс, который указывает, что это удаленный объект. Так объекты класса, выполняющие поиск (Lookup), отмечены удаленными ссылками. Все методы в удаленном интерфейсе должны быть объявлены через исключение типа Java.rmi.RemoteException, которое активизируется всякий раз, когда метод удаленного вызова дает сбои.

Разработка серверного кода

После того как вы определили интерфейс к удаленному объекту, нужно выполнить следующий шаг — разработать код сервера. Сервер осуществляет объектный интерфейс и создает образцы объекта, который будет дистанционно разделяем.

Для нашего примера это будет выглядеть следующим образом:

```
  // LookupServer.java

import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;

public class LookupServer extends UnicastRemoteObject
                        implements Lookup

private Vector save = new Vector();

 public LookupServer(String db) throws RemoteException
    {
       try {
          FileReader fr = new FileReader(db);
          BufferedReader br = new BufferedReader(fr);
          String s = null;
          while ((s = br.readLine()) != null)
             save.addElement(s);
          fr.close();
       }
       catch (Throwable e) {
          System.err.println("exception");
          System.exit(1);
       }
    }

     public String findInfo(String info)
     {
```

```
        if (info == null)
        return null;

        info = info.toLowerCase();
        int n = save.size();
        for (int i = 0; i < n; i++) {
           String dbs = (String)save.elementAt(i);
           if (dbs.toLowerCase().indexOf(info) != -1)
           return dbs;
        }

        return null;
     }

     public static void main(String args[])
     {
        try {
           RMISecurityManager security =
                         new RMISecurityManager();
           System.setSecurityManager(security);
           String db = args[0];
           LookupServer server = new LookupServer(db);
           Naming.rebind("LookupServer", server);
           System.err.println("LookupServer ready...");
        }
        catch (Throwable e) {
           System.err.println("exception: " + e);
           System.exit(1);
        }
     }
  }
```

Сервер читает в текстовой базе номера телефонов и имена и сохраняет их. Метод `findInfo` ищет затем нужное имя и телефон.

Пример базы данных

```
Ivanov, Ivan 295-0083
Petrov, Peter 775-9958
Romanov, Alexander 555-7779
```

Заметим, что `LookupServer` является расширением стандартного класса `java.rmi.server.UnicastRemoteObject` и выполняет поиск (`Lookup`). Один из этих классов обеспечивает некоторые базисные реквизиты, необходимые для удаленных объектов, а другой определяет методы, которые будут вызваны дистанционно.

Установка службы безопасности

Наиболее сложная часть этого кода — то, что происходит в процедуре `main()`. Первым делом нужно установить защиту. RMI принужден загружать удаленные файлы типа class, и в этом смысле напоминает какой-нибудь Web-браузер с его операциями по загрузке апплетов, что само по себе всегда небезопасно. Если вы не установили защиту, то по умолчанию должны загружаться только локальные файлы, и RMI по определению не может работать с такими ограничениями. Так что вы должны установить `security manager`, чтобы сделать возможной загрузку удаленных class-файлов.

Образец `LookupServer` затем регистрируется с помощью службы `Naming.rebind` и становится доступным клиенту по имени.

Вы могли бы задаться вопросом, как удаленный метод фактически становится вызываемым, если сервер не содержит никакого сетевого кода и никаких TCP/IP-примитивов? Это происходит за сценой, поскольку сервер и клиент используют так называемые *скелетоны* и *стабы* для коммуникации между собой. Соответствующие class-файлы генерируются из серверного class-файла через RMI-транслятор, описанный ниже.

Концептуально, класс stub (заглушка) выглядит так:

```
public class LookupServer_Stub extends java.rmi.server.RemoteStub
                          implements Lookup, java.rmi.Remote { ...  }
```

а скелетон — так:

```
public class LookupServer_Skel implements
                               java.rmi.server.Skeleton { ...  }
```

Использование команды:

```
javap -c LookupServer_Stub
```

будет показывать байт-код и иллюстрировать то, что происходит за сценой.

Стаб (stub) — это суррогат для удаленного объекта, а скелетон — некая сущность на сервере, которая обрабатывает удаленные вызовы.

Стаб обеспечивает функции приема-передачи на стороне клиента, а скелетон — на стороне сервера. При этом производится преобразование объектов в последовательную форму, а проще говоря, в поток байтов, передаваемых с помощью протокола TCP/IP.

Разработка клиентского кода

```
// LookupClient.java

import java.rmi.*;
import java.rmi.server.*;

public class LookupClient {
   public static void main(String args[])
```

```
    {
     try {
       RMISecurityManager security =
                        new RMISecurityManager();
       System.setSecurityManager(security);
       String host = "localhost";
       String server = "LookupServer";
       String name = "rmi://" + host + "/" + server;
        Lookup look_obj = (Lookup)Naming.lookup(name);
          String results = look_obj.findInfo(args[0]);
          if (results == null)
             System.err.println("** not found **");
          else
             System.out.println(results);
       }
       catch (Throwable e) {
          System.err.println("exception: " + e);
          System.exit(1);
       }
    }
}
```

Если только вы активизируете сервер, выполняя прямой клиентский запрос, то защита для пользователя определяется такая же, как и на сервере. URL при этом определяется как:

```
Rmi://localhost/LookupServer
```

где `localhost` — имя локального компьютера (IP-адрес 127.0.0.1), используемого как сервер. Клиент располагается на той же самой машине. Вы можете также использовать и удаленную главную ЭВМ. Когда вызов к нужному методу сделан — результаты незамедлительно передаются клиенту.

Компиляция кода

Три файла — Lookup.java, LookupServer.java и LookupClient.java — компилируются, как и обычно, в Java:

```
javac Lookup.java
javac LookupServer.java
javac LookupClient.java
```

Выполнение RMI-компилятора

После того как вы откомпилируете эти файлы, выполните команду RMI Compiler (rmic):

```
rmic LookupServer
```

чтобы получить файлы LookupServer_Skel.class и LookupServer_Stub.class.

Перемещение class-файлов в соответствующее им место на диске

Вы должны переместить клиентские файлы (Lookup.class, LookupClient.class и LookupServer_Stub.class) в каталог, откуда вы, как клиент, будете их выполнять.

Вы должны переместить серверные файлы (Lookup.class, LookupServer.class, LookupServer_Skel.class и LookupServer_Stub.class) в каталог, где они станут доступными для публичного доступа.

Регистрация с помощью Rmiregistry

Объект, к которому обращаются дистанционно, должен быть введен в запись объектов, т. е. зарегистрирован. В JDK 1.1 имеется специальная программа Rmiregistry.

Rmiregistry может выполняться в отдельном окне или как фоновый процесс на сервере.

Старт сервера

Вы должны стартовать сервер по команде:

```
java LookupServer database_name
```

К примеру, если ваша база данных с набором имен и телефонов находится в файле C:\PHONE.TXT, то вы должны дать команду:

```
java LookupServer C:\PHONE.TXT
```

Старт клиента

Клиентская программа стартуется по команде:

```
java LookupClient Ivanov
```

где `Ivanov` — это искомое имя в базе.

Таким образом, RMI дает возможность создавать распределенные прикладные программы Java-to-Java, в которых методы удаленных объектов Java вызываются из других Java-программ на различных главных ЭВМ так, как если бы эти методы вызывались локально. Естественно, что подобные возможности можно эффективно использовать при работе с SQL-серверами, которые предоставляют интерфейс Java-API для доступа к данным. Например, в группе продуктов Informix в Informix Client SDK дано описание *Informix Object Interface for Java*, где приводятся многочисленные примеры, как организовать взаимодействующие RMI-приложения с доступом к базам данных Informix. Более того, имеется и соответствующий RMI-сервер, который содержит массу удобных и полезных методов, которые можно вызывать дистанционно. В сущности, вышеприведенный пример можно приспособить для работы с любыми базами данных и SQL-серверами, если вы знаете, каким образом устроен интерфейс Java-API для доступа к базам данных. В крайнем случае, не возбраняется и использование JDBC в RMI-приложениях,

хотя вряд ли это будет в достаточной степени эффективно. В Informix, например, для непосредственного взаимодействия с базами данных существуют два RMI-пакета: informix.api.remote.rmi — для удаленных клиентов и informix.api.remote.rmi.server — для RMI-сервера. При этом в клиентском приложении используется интерфейс к DBMSManager, который накапливает информацию обо всех серверах Informix и базах данных, и вы можете установить либо локальное, либо удаленное соединение с базой через RMI-сервер. Для локального соединения создается объект `DirectDBMSManager`. Для удаленного соединения создается объект `RMIDBMS-Manager` и передается к соответствующему RMI-серверу. Спецификация RMI-сервера осуществляется в форме:

```
rmi://hostname[:port]/
```

```
// Создание объекта DBMSManager
DBMSManager getDBMSManager() throws Exception
{
// based on RMI Checkbox, get appropriate DBMSManager
DBMSManager dbmsManager;
if (RMIcheckbox.getState())
  dbmsManager = new RMIDBMSManager(rmiServerTextField.getText());
else
  dbmsManager = new DirectDBMSManager();
return dbmsManager;
```

Взаимодействие с базой осуществляется с помощью специальных транзакционных методов и методов управления курсором, которые могут вызываться как локально, так и дистанционно. В различных базах и SQL-серверах осуществляется по-разному, но если сервер кроссплатформенный — единый и универсальный интерфейс Java-API играет немаловажную роль.

Инкапсулированный в СУБД Java

Java может быть встроен в СУБД множеством различных способов и при этом всегда достигается решение сразу нескольких задач.

- **Применение полноценного языка программирования для написания хранимых процедур.** В сервер встраивается виртуальная машина Java и внутренний интерфейс JDBC. Встраивание мощного и безопасного языка программирования позволяет обойти ограничения хранимых процедур на языке SQL.
- **Расширение объектных типов данных.** Объекты, написанные на языке Java, могут храниться в виде значений в реляционной таблице. Это позволяет создавать и использовать произвольные типы данных. Java-класс может быть использован в качестве типа данных для столбца таблицы. Каждое поле такой записи становится экземпляром соответствующего

Java-класса. В качестве примера возьмем простой класс на языке Java, хранящий адреса. Прикладной код может быть включен в методы класса. Например, строковые данные могут содержаться только в полях `Street` (улица) и `Postal Code` (почтовый код), а значение поля `City` (город) может вычисляться на основе почтового индекса. Более того, можно использовать наследование классов и методов. При этом допускается перегрузка методов в зависимости от используемого класса. Так можно создать классы `US_Address` и `Rus_Address`, унаследованные от базового класса `Address`. В каждом таком классе могут быть разные методы определения городов по почтовому индексу и методы проверки его значений, естественно, разные для России и США. Даже SQL может использоваться для доступа к Java-объектам, как это уже сделано в Sybase Adaptive Server.

Следующий пример вставляет новую запись в таблицу:

```
INSERT INTO employees (id, name, Address)
VALUES(1789,
   "Serg Dunaev",
    new Rus_Address("58 Gagarin Street", "153038")
)
```

Приведенный ниже запрос с использованием Java-объектов возвращает список сотрудников, живущих на указанной улице:

```
SELECT name
FROM employees
WHERE Rus_Address.street="Gagarin Street"
```

- **Единая программная модель.** Впервые прикладные программные компоненты можно будет перемещать между клиентскими программами, серверами приложений и СУБД. Разработчики смогут использовать единую программную модель на всех уровнях информационной системы. Произвольная программа на Java состоит из набора классов. Для их использования необходимо провести инсталляцию Java-классов в СУБД. Классы должны быть откомпилированы в байт-код и, тем самым, готовы для использования в любой виртуальной машине. В связи с применением виртуальной машины для выполнения методов на Java и встроенной поддержке интерфейса JDBC, один и тот же объект на языке Java может быть использован как внутри, так и вне СУБД.

Новые возможности работы с таблицами с использованием JFC-классов

В версии JDK1.2, которую ждали так долго и которая появилась в декабре 1998 года, существует много новых удобных возможностей для организации эффективной работы с базами данных. В частности, к ним относится так

называемая *табличная модель* (Table Model), которая входит в набор дополнительных Java Foundation Classes (JFC), ранее известной под именем Swing. JFC обеспечивает новый набор компонентов ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ с расширенными возможностями. JFC выполнен на 100 % чистом языке Java и основан на JDK 1.1. При помощи JFC можно разрабатывать великолепно оформленные графические компоненты и интерфейсы для любой платформы OS (Windows, Solaris, Macintosh). В составе многочисленных JFC-классов имеется класс `javax.swing.table`, который предлагает разнообразные методы работы с таблицами В самом деле, автоматизировать работу с различными таблицами представляется весьма важной и актуальной задачей. В вышеупомянутом классе главную роль играет интерфейс Table Model, с помощью которого можно избавить себя от программирования рутинных процедур определения ширины и количества столбцов таблицы, разлиновки клеток и т. д. Более того, можно написать несколько специальных классов вроде сортировки выбранных столбцов таблицы или именования колонок, которые можно без всяких изменений использовать с любыми таблицами. Ниже приводятся несколько программ на Java, предназначенных для распечатки на экране дисплея любых таблиц, где бы они не находились — на сервере баз данных или в отдельных файлах специального формата. Для доступа к таблицам эти программы используют стандартный интерфейс JDBC.

Первая программа TableExample1.java вводит в режиме командной строки все необходимые параметры для связи.

- URL-базы данных (например, `jdbc:odbc:table1_on_ultra1`).
- Тип используемого JDBC-драйвера (например, `sun.jdbc.odbc.JdbcOdbcDriver`).
- Имя пользователя (например, `Informix`).
- Пароль.
- Запрос к базе на SQL (например, `select * from object`)

```
Java TableExample1 "jdbc:odbc:table1_on_ultra1" "sun.jdbc.JdbcOdbcDriver"
"Informiz" "xyz" "select * from object"
```

Вторая программа TableExample2.java позволяет вводить те же параметры в режиме диалога в специальном окне и дополнена некоторыми элементами меню и графическим интерфейсом с пользователем.

Обе программы нуждаются в одних и тех же дополнительных модулях.

JDBCAdapter.java — эта программа непосредственно реализует интерфейс Table Model на основе JFC-классов, добавляет столбцы и определяет их тип на основе метамодели таблицы, заполняет их данными, выполняя все необходимые преобразования, и строит таблицу.

Программа TableMap.java обрабатывает все события и перенаправляет их к Table Model, работает в режиме "прослушивания" и обрабатывает инициативы всех входящих запросов.

Программа TableSorter.java — класс, который сортирует столбцы таблицы по нажатию кнопки мыши при выборе соответствующего заголовка столбца. Поскольку, какой именно столбец выбрать в качестве основного параметра сортировки, решает пользователь, то класс TableSorter наследует класс TableMap для обработки события нажатия мыши.

Пример 1. Простая программа TableExample1 для извлечения и просмотра таблицы с сервера Informix.

```
// Copyright (c) 1997–1999 by Sun Microsystems, Inc. All Rights Reserved
import javax.swing.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.Dimension;

public class TableExample1 {

public TableExample1(String URL, String driver, String user,
                     String passwd, String query) {
   JFrame frame = new JFrame("Table");
   frame.addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent e) {System.exit(0);}});
      JDBCAdapter dt = new JDBCAdapter(URL, driver, user, passwd);
      dt.executeQuery(query);

      // Создание таблицы
      JTable tableView = new JTable(dt);

      JScrollPane scrollpane = new JScrollPane(tableView);
      scrollpane.setPreferredSize(new Dimension(700, 300));

      frame.getContentPane().add(scrollpane);
      frame.pack();
      frame.setVisible(true);
   }
   public static void main(String[] args) {
      if (args.length != 5) {
      System.err.println("Needs database parameters eg. ...");
      System.err.println("java TableExample1 \"jdbc:odbc:stat_on_ultral\"
\"sun.jdbc.odbc.JdbcOdbcDriver\" \"informix\" \"Xtvjlfy7\" \"select *
from object\"");
         return;
      }
   new TableExample1(args[0], args[1], args[2], args[3], args[4]);
   }
}
```

Пример 2. Программа TableExample2 использует графический интерфейс JFC и диалог с пользователем для извлечения и просмотра таблицы с SQL-

сервера. (Пример взят из стандартного дистрибутива JDK 1.2, распространяемого фирмой Sun Microsystems.)

```
// Автор Филипп Милн (Philip Milne)
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;
import javax.swing.border.*;

public class TableExample2 implements LayoutManager {
   static String[] ConnectOptionNames = { "Соединение" };
   static String   ConnectTitle = "Информация о соединении";

   Dimension   origin = new Dimension(0, 0);

   JButton     fetchButton;
   JButton     showConnectionInfoButton;

   JPanel      connectionPanel;
   JFrame      frame; // The query/results window.

   JLabel      userNameLabel;
   JTextField  userNameField;
   JLabel      passwordLabel;
   JTextField  passwordField;
   // JLabel      queryLabel;
   JTextArea   queryTextArea;
   JComponent  queryAggregate;
   JLabel      serverLabel;
   JTextField  serverField;
   JLabel      driverLabel;
   JTextField  driverField;

   JPanel      mainPanel;

   TableSorter sorter;
   JDBCAdapter dataBase;
   JScrollPane tableAggregate;

   public TableExample2() {
      mainPanel = new JPanel();

      // Создать панель для информации о соединении
   createConnectionDialog();

   // Создать кнопку управления.
   showConnectionInfoButton = new JButton("Кофигурация");
```

```
 showConnectionInfoButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            activateConnectionDialog();
        }
    }
);

fetchButton = new JButton("Загрузка");
   fetchButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            fetch();
        }
    }
);

// Создать область для текстового запроса и метку.
   queryTextArea = new JTextArea("SELECT * FROM object", 25, 25);
queryAggregate = new JScrollPane(queryTextArea);
  queryAggregate.setBorder(new BevelBorder(BevelBorder.LOWERED));

  // Создать таблицу.
  tableAggregate = createTable();
  tableAggregate.setBorder(new BevelBorder(BevelBorder.LOWERED));

// Добавить все компоненты к главной панели.
   mainPanel.add(fetchButton);
   mainPanel.add(showConnectionInfoButton);
   mainPanel.add(queryAggregate);
   mainPanel.add(tableAggregate);
   mainPanel.setLayout(this);

   // Создать рамку окна и разместить ее на главной панели.
   frame = new JFrame("TableExample2");
   frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}});
   frame.setBackground(Color.lightGray);
   frame.getContentPane().add(mainPanel);
   frame.pack();
   frame.setVisible(false);
   frame.setBounds(200, 200, 640, 480);

activateConnectionDialog();
}
/**
 * Активизируй JDialog, используя JOptionPanel, содержащую
 * connectionPanel.
 * Если пользователь щелкнет кнопку Connect, соединение сбросится.
 */
void activateConnectionDialog() {
if(JOptionPane.showOptionDialog(tableAggregate, connectionPanel,
   ConnectTitle,
```

```
JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE,
        null, ConnectOptionNames, ConnectOptionNames[0]) == 0) {
    connect();
      frame.setVisible(true);
}
else if(!frame.isVisible())
    System.exit(0);
}
public void addLayoutComponent(String s, Component c) {}
public void connect() {
   dataBase = new JDBCAdapter(
      serverField.getText(),
      driverField.getText(),
      userNameField.getText(),
      passwordField.getText());
   sorter.setModel(dataBase);
}
/**
 * Создадим connectionPanel, которая будет содержать все поля
 * информации о соединении.
 */
public void createConnectionDialog() {
// Create the labels and text fields.
userNameLabel = new JLabel("Имя пользователя: ", JLabel.RIGHT);
userNameField = new JTextField("guest");

passwordLabel = new JLabel("Пароль: ", JLabel.RIGHT);
passwordField = new JTextField("trustworthy");

   serverLabel = new JLabel("Database URL: ", JLabel.RIGHT);
serverField = new JTextField("jdbc:odbc:stat_on_ultra1");

driverLabel = new JLabel("Driver: ", JLabel.RIGHT);
driverField = new JTextField("sun.jdbc.odbc.JdbcOdbcDriver");

connectionPanel = new JPanel(false);
connectionPanel.setLayout(new BoxLayout(connectionPanel,
                  BoxLayout.X_AXIS));

JPanel namePanel = new JPanel(false);
namePanel.setLayout(new GridLayout(0, 1));
namePanel.add(userNameLabel);
namePanel.add(passwordLabel);
namePanel.add(serverLabel);
namePanel.add(driverLabel);

JPanel fieldPanel = new JPanel(false);
fieldPanel.setLayout(new GridLayout(0, 1));
```

```
   fieldPanel.add(userNameField);
   fieldPanel.add(passwordField);
   fieldPanel.add(serverField);
      fieldPanel.add(driverField);

   connectionPanel.add(namePanel);
   connectionPanel.add(fieldPanel);
   }
   public JScrollPane createTable() {
      sorter = new TableSorter();

      //connect();
      //fetch();

      // Создать таблицу
      JTable tableView = new JTable(sorter);

   // Инсталлировать обработчик событий "мыши" в TableHeader как
   // указатель сортировки.

      sorter.addMouseListenerToHeaderInTable(tableView);
      JScrollPane scrollpane = new JScrollPane(tableView);
      return scrollpane;
   }
   public void fetch() {
      dataBase.executeQuery(queryTextArea.getText());
   }
   public void layoutContainer(Container c) {
      Rectangle b = c.getBounds();
      int topHeight = 90;
      int inset = 4;
      showConnectionInfoButton.setBounds(b.width-2*inset-120,
                                         inset, 120, 25);
      fetchButton.setBounds(b.width-2*inset-120, 60, 120, 25);
      // queryLabel.setBounds(10, 10, 100, 25);
      queryAggregate.setBounds(inset, inset, b.width-2*inset - 150, 80);
      tableAggregate.setBounds(new Rectangle(inset, inset + topHeight,
         b.width-2*inset,
         b.height-2*inset - topHeight));
   }
   public static void main(String s[]) {
      new TableExample2();
   }
   public Dimension minimumLayoutSize(Container c){return origin;}
   public Dimension preferredLayoutSize(Container c){return origin;}
   public void removeLayoutComponent(Component c) {}
}
```

Далее приведены общие модули.

Модуль JDBCAdapter.java

```
/**
 * Адаптор осуществляет перевод JDBC-интерфейса в TableModel-интерфейс.
 *
 * Версия 1.20 09/25/97
 * Автор Филипп Милн
 */
import java.util.Vector;
import java.sql.*;
import javax.swing.table.AbstractTableModel;
import javax.swing.event.TableModelEvent;

public class JDBCAdapter extends AbstractTableModel {
    Connection          connection;
    Statement           statement;
    ResultSet           resultSet;
    String[]            columnNames = {};
    Vector            rows = new Vector();
    ResultSetMetaData   metaData;

    public JDBCAdapter(String url, String driverName,
                       String user, String passwd) {
        try {
            Class.forName(driverName);
            System.out.println("Открытие соединения с базой данных");

            connection = DriverManager.getConnection(url, user, passwd);
            statement = connection.createStatement();
        }
        catch (ClassNotFoundException ex) {
            System.err.println("Невозможно найти драйвер базы данных.");
            System.err.println(ex);
        }
        catch (SQLException ex) {
            System.err.println("Невозможно установить соединение с этой
            базой данных.");
            System.err.println(ex);
        }
     }

    public void executeQuery(String query) {
        if (connection == null || statement == null) {
            System.err.println("Запрос к базе нельзя выполнить.");
            return;
        }
```

```
        try {
            resultSet = statement.executeQuery(query);
            metaData = resultSet.getMetaData();

            int numberOfColumns =  metaData.getColumnCount();
            columnNames = new String[numberOfColumns];
            // Получить имена колонок и кэшировать их.
            // Теперь можно закрыть соединение.
            for(int column = 0; column < numberOfColumns; column++) {
                columnNames[column] = metaData.getColumnLabel(column+1);
            }

            // Считать все строки.
            rows = new Vector();
            while (resultSet.next()) {
                Vector newRow = new Vector();
                for (int i = 1; i <= getColumnCount(); i++) {
                   newRow.addElement(resultSet.getObject(i));
                }
                rows.addElement(newRow);
            }
            //  close(); Необходимо скопировать metaData, т. к. имеется
            //  ошибка в jdbc:odbc driver.
            fireTableChanged(null); // Сообщить обработчику (listeners)
            // о новой таблице.
        }
        catch (SQLException ex) {
            System.err.println(ex);
        }
    }

    public void close() throws SQLException {
        System.out.println("Закрыть соединение с базой");
        resultSet.close();
        statement.close();
        connection.close();
    }

    protected void finalize() throws Throwable {
        close();
        super.finalize();
    }

////////////////////////////////////////////////////////////////////////////
    //
    //              Реализация TableModel-интерфейса
    //
////////////////////////////////////////////////////////////////////////////
```

```
// MetaData

public String getColumnName(int column) {
    if (columnNames[column] != null) {
        return columnNames[column];
    } else {
        return "";
    }
}

public Class getColumnClass(int column) {
    int type;
    try {
        type = metaData.getColumnType(column+1);
    }
    catch (SQLException e) {
        return super.getColumnClass(column);
    }

    switch(type) {
    case Types.CHAR:
    case Types.VARCHAR:
    case Types.LONGVARCHAR:
        return String.class;

    case Types.BIT:
        return Boolean.class;

    case Types.TINYINT:
    case Types.SMALLINT:
    case Types.INTEGER:
        return Integer.class;

    case Types.BIGINT:
        return Long.class;

    case Types.FLOAT:
    case Types.DOUBLE:
        return Double.class;

    case Types.DATE:
        return java.sql.Date.class;

    default:
        return Object.class;
    }
}

public boolean isCellEditable(int row, int column) {
    try {
```

```
            return metaData.isWritable(column+1);
        }
        catch (SQLException e) {
            return false;
        }
    }

    public int getColumnCount() {
        return columnNames.length;
    }

    // Data methods

    public int getRowCount() {
        return rows.size();
    }

    public Object getValueAt(int aRow, int aColumn) {
        Vector row = (Vector)rows.elementAt(aRow);
        return row.elementAt(aColumn);
    }

    public String dbRepresentation(int column, Object value) {
        int type;

        if (value == null) {
            return "null";
        }

        try {
            type = metaData.getColumnType(column+1);
        }
        catch (SQLException e) {
            return value.toString();
        }

        switch(type) {
        case Types.INTEGER:
        case Types.DOUBLE:
        case Types.FLOAT:
            return value.toString();
        case Types.BIT:
            return ((Boolean)value).booleanValue() ? "1" : "0";
        case Types.DATE:
            return value.toString(); // Здесь необходимо преобразование.
        default:
            return "\""+value.toString()+"\"";
        }

    }
```

```
    public void setValueAt(Object value, int row, int column) {
        try {
            String tableName = metaData.getTableName(column+1);
            // Some of the drivers seem buggy, tableName не должна быть
               нулевой
            if (tableName == null) {
                System.out.println("Table name returned null.");
            }
            String columnName = getColumnName(column);
            String query =
                "update "+tableName+
                " set "+columnName+" = "+dbRepresentation(column, value)+
                " where ";
            // Будем блокировать все подряд, чтобы
            // обеспечить возможность редактирования
            //
            for(int col = 0; col<getColumnCount(); col++) {
                String colName = getColumnName(col);
                if (colName.equals("")) {
                    continue;
                }
                if (col != 0) {
                    query = query + " and ";
                }
                query = query + colName +" = "+
                    dbRepresentation(col, getValueAt(row, col));
            }
            System.out.println(query);
            System.out.println("Not sending update to database");
            // statement.executeQuery(query);
        }
        catch (SQLException e) {
            //     e.printStackTrace();
            System.err.println("Update failed");
        }
        Vector dataRow = (Vector)rows.elementAt(row);
        dataRow.setElementAt(value, column);
```

Модуль TableMap.java

```
/*
 * В цепочке манипуляторов имеются некоторые общие методы и свойства
 * (behaviour). TableMap обеспечивает большинство этих свойств, хотя
* на него могут накладываться фильтры, которые могут отменить
* специфические методы. TableMap обеспечивает выполнение TableModel,
```

```
* направляя все запросы к модели, а TableModelListener обрабатывает
* все события, обслуживая асинхронные подпрограммы обработки
* входящих запросов. TableMap, который не включен в цепочку
* фильтров таблицы, не должен иметь никакого эффекта.
 *
 * Версия 1.6 08/26/98
 * Автор Филипп Милн */

import javax.swing.table.*;
import javax.swing.event.TableModelListener;
import javax.swing.event.TableModelEvent;

public class TableMap extends AbstractTableModel implements
TableModelListener
{
    protected TableModel model;

    public TableModel  getModel() {
        return model;
    }

    public void  setModel(TableModel model) {
        this.model = model;
        model.addTableModelListener(this);
    }

    // По умолчанию TableModel реализуется пересылкой всех сообщений
    // к модели.

    public Object getValueAt(int aRow, int aColumn) {
        return model.getValueAt(aRow, aColumn);
    }

    public void setValueAt(Object aValue, int aRow, int aColumn) {
        model.setValueAt(aValue, aRow, aColumn);
    }

    public int getRowCount() {
        return (model == null) ? 0 : model.getRowCount();
    }

    public int getColumnCount() {
        return (model == null) ? 0 : model.getColumnCount();
    }

    public String getColumnName(int aColumn) {
        return model.getColumnName(aColumn);
    }
```

```
    public Class getColumnClass(int aColumn) {
        return model.getColumnClass(aColumn);
    }

    public boolean isCellEditable(int row, int column) {
         return model.isCellEditable(row, column);
    }
//
// Выполнение интерфейса TableModelListener
//

    // По умолчанию передать все события обработчикам событий.
    public void tableChanged(TableModelEvent e) {
        fireTableChanged(e);
    }
}
```

Модуль сортировки TableSorter.java

```
/**
 * Сортировщик для TableModels. У сортировщика своя модель
 * (соответствующая TableModel)
 * и он сам выполняет TableModel. TableSorter не сохраняет и не копирует
 * данные в TableModel, вместо этого он управляет массивом чисел,
 * которые сохраняются с тем же размером как числа в строках модели таблицы.
 * Когда модель изменяется, она извещает сортировщика, что произошло
 * изменение, например, "rowsAdded", так что внутренний массив чисел может
 * быть перезагружен. Все запросы, сделанные сортировщиком (подобные
 * getValueAt(row, col), переадресуются к модели через "мапирование"
 * массива. Таким способом TableSorter может обработать другую копию таблицы,
 * в которой строки расположены в другом порядке.
 *
 * Версия 1.7 08/26/98
 * Автор Филип Милн
 */

import java.util.*;

import javax.swing.table.TableModel;
import javax.swing.event.TableModelEvent;

// Импорт необходимых классов для обработки событий "мыши" из JTable.

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.InputEvent;
```

```
import javax.swing.JTable;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableColumn;
import javax.swing.table.TableColumnModel;

public class TableSorter extends TableMap
{
    int             indexes[];
    Vector          sortingColumns = new Vector();
    boolean         ascending = true;
    int compares;

    public TableSorter()
    {
        indexes = new int[0]; // For consistency.
    }

    public TableSorter(TableModel model)
    {
        setModel(model);
    }

    public void setModel(TableModel model) {
        super.setModel(model);
        reallocateIndexes();
    }

    public int compareRowsByColumn(int row1, int row2, int column)
    {
        Class type = model.getColumnClass(column);
        TableModel data = model;

        // Check for nulls

        Object o1 = data.getValueAt(row1, column);
        Object o2 = data.getValueAt(row2, column);

        // If both values are null return 0
        if (o1 == null && o2 == null) {
            return 0;
        }
        else if (o1 == null) { // Define null less than everything.
            return -1;
        }
        else if (o2 == null) {
            return 1;
        }

/* Мы копируем все возвращаемые значения из вызова getValue в случае, если
оптимизированная модель многократно использует один объект, чтобы возвратить
множество значений.
```

```
Подклассы Number из JDK не будут использоваться в этом случае, но другие
подклассы Number могли бы сделать это, чтобы сохранить память и избежать ее
ненужного перераспределения.
*/
        if (type.getSuperclass() == java.lang.Number.class)
            {
                Number n1 = (Number)data.getValueAt(row1, column);
                double d1 = n1.doubleValue();
                Number n2 = (Number)data.getValueAt(row2, column);
                double d2 = n2.doubleValue();

                if (d1 < d2)
                    return -1;
                else if (d1 > d2)
                    return 1;
                else
                    return 0;
            }
        else if (type == java.util.Date.class)
            {
                Date d1 = (Date)data.getValueAt(row1, column);
                long n1 = d1.getTime();
                Date d2 = (Date)data.getValueAt(row2, column);
                long n2 = d2.getTime();

                if (n1 < n2)
                    return -1;
                else if (n1 > n2)
                    return 1;
                else return 0;
            }
        else if (type == String.class)
            {
                String s1 = (String)data.getValueAt(row1, column);
                String s2    = (String)data.getValueAt(row2, column);
                int result = s1.compareTo(s2);

                if (result < 0)
                    return -1;
                else if (result > 0)
                    return 1;
                else return 0;
            }
        else if (type == Boolean.class)
            {
                Boolean bool1 = (Boolean)data.getValueAt(row1, column);
                boolean b1 = bool1.booleanValue();
```

```
                Boolean bool2 = (Boolean)data.getValueAt(row2, column);
                boolean b2 = bool2.booleanValue();

                if (bl == b2)
                    return 0;
                else if (bl) // Define false < true
                    return 1;
                else
                    return -1;
            }
        else
            {
                Object v1 = data.getValueAt(row1, column);
                String s1 = v1.toString();
                Object v2 = data.getValueAt(row2, column);
                String s2 = v2.toString();
                int result = s1.compareTo(s2);

                if (result < 0)
                    return -1;
                else if (result > 0)
                    return 1;
                else return 0;
            }
    }

    public int compare(int row1, int row2)
    {
        compares++;
        for(int level = 0; level < sortingColumns.size(); level++)
            {
                Integer column = (Integer)sortingColumns.elementAt(level);
                int result = compareRowsByColumn(row1, row2,
column.intValue());
                if (result != 0)
                    return ascending ? result : -result;
            }
        return 0;
    }

    public void  reallocateIndexes()
    {
        int rowCount = model.getRowCount();

        // Установить новый массив индексов с нужным числом элементов
        // для новой модели данных .
        indexes = new int[rowCount];

        // Initialise with the identity mapping.
        for(int row = 0; row < rowCount; row++)
```

```
        indexes[row] = row;
    }

    public void tableChanged(TableModelEvent e)
    {
       System.out.println("Sorter: tableChanged");
        reallocateIndexes();

        super.tableChanged(e);
    }

    public void checkModel()
    {
        if (indexes.length != model.getRowCount()) {
            System.err.println("Сортировщик не информирован об изменениях
            в модели.");
        }
    }

    public void  sort(Object sender)
    {
        checkModel();

        compares = 0;
        // n2sort();
        // qsort(0, indexes.length-1);
        shuttlesort((int[])indexes.clone(), indexes, 0, indexes.length);
        System.out.println("Compares: "+compares);
    }

    public void n2sort() {
        for(int i = 0; i < getRowCount(); i++) {
            for(int j = i+1; j < getRowCount(); j++) {
                if (compare(indexes[i], indexes[j]) == -1) {
                    swap(i, j);
                }
            }
        }
    }

    // Это сугубо внутренняя реализация, которую мы не имели времени
    // исследовать и которая может работать плохо в некоторых
    // обстоятельствах.
    // Она требует двойного пространства для встроенного алгоритма.

    public void shuttlesort(int from[], int to[], int low, int high) {
        if (high - low < 2) {
            return;
        }
        int middle = (low + high)/2;
```

```
        shuttlesort(to, from, low, middle);
        shuttlesort(to, from, middle, high);

        int p = low;
        int q = middle;

        /* Это — дополнительный трюк; в каждом рекурсивном вызове надо
           проверить,
           нет ли уже упорядоченного элемента в этом поднаборе. Если это
           так, то
           никакие дальнейшие сравнения не являются необходимыми;
           Когда число элементов — три, они разбиваются на разделы так,
           чтобы первый набор, (low,mid), имел один элемент
           и второй, (mid, high),
           имел два. Мы проскочим оптимизацию, когда число элементов -
           три или
           меньше, т. к. первый элемент сравнения в нормальном слиянии
           произведет ту же самую последовательность шагов. Эта
           оптимизация
           заслуживает внимания для частично упорядоченных списков,
           но при этом
           необходим некоторый анализ, чтобы найти, как понизится
           эффективность к
           Nlog (N), если начальный порядок задан на убывание — она
           может понижаться очень быстро. */

        if (high — low >= 4 && compare(from[middle-1], from[middle]) <= 0) {
            for (int i = low; i < high; i++) {
                to[i] = from[i];
            }
            return;
        }

        // Нормальное слияние.

        for(int i = low; i < high; i++) {
            if (q >= high || (p < middle && compare(from[p], from[q]) <= 0))
{
                to[i] = from[p++];
            }
            else {
                to[i] = from[q++];
            }
        }
    }

    public void swap(int i, int j) {
        int tmp = indexes[i];
        indexes[i] = indexes[j];
```

```
        indexes[j] = tmp;
    }

    // Установка соответствия влияет только на строки данных.
    // Передача всех запросов к этим строкам происходит через массив: "indexes".

    public Object getValueAt(int aRow, int aColumn)
    {
        checkModel();
        return model.getValueAt(indexes[aRow], aColumn);
    }

    public void setValueAt(Object aValue, int aRow, int aColumn)
    {
        checkModel();
        model.setValueAt(aValue, indexes[aRow], aColumn);
    }

    public void sortByColumn(int column) {
        sortByColumn(column, true);
    }

    public void sortByColumn(int column, boolean ascending) {
        this.ascending = ascending;
        sortingColumns.removeAllElements();
        sortingColumns.addElement(new Integer(column));
        sort(this);
        super.tableChanged(new TableModelEvent(this));
    }

    //
    // Добавьте MouseListener к Table, чтобы начать сортировку таблицы,
    // когда выбран и нажат столбец заголовка в JTable.

    public void addMouseListenerToHeaderInTable(JTable table) {
        final TableSorter sorter = this;
        final JTable tableView = table;
        tableView.setColumnSelectionAllowed(false);
        MouseAdapter listMouseListener = new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                TableColumnModel columnModel = tableView.getColumnModel();
                int viewColumn = columnModel.getColumnIndexAtX(e.getX());
                int column =
tableView.convertColumnIndexToModel(viewColumn);
                if(e.getClickCount() == 1 && column != -1) {
                    System.out.println("Sorting ...");
                    int shiftPressed =
e.getModifiers()&InputEvent.SHIFT_MASK;
                    boolean ascending = (shiftPressed == 0);
                    sorter.sortByColumn(column, ascending);
```

```
                }
            }
        };
        JTableHeader th = tableView.getTableHeader();
        th.addMouseListener(listMouseListener);
    }
}
```

Результат компиляции и запуска программы TableExample2 изображен на рис. 6.8.

На рис. 6.9 изображено окно из программы для ввода SQL-запроса.

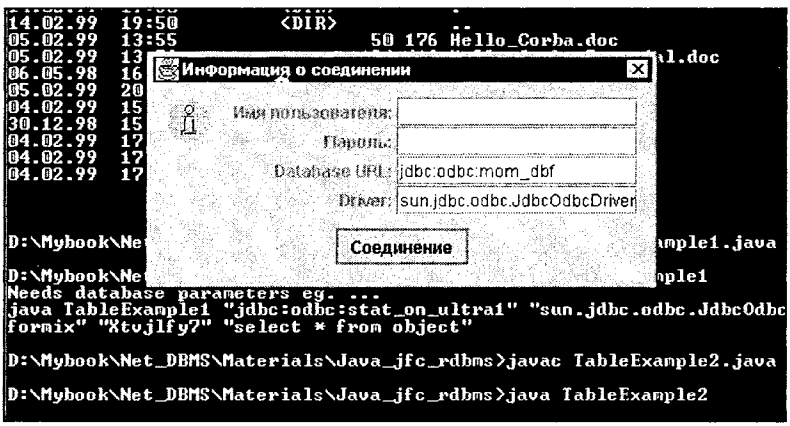


Рис. 6.8. Компиляция и запуск программы TableExample2

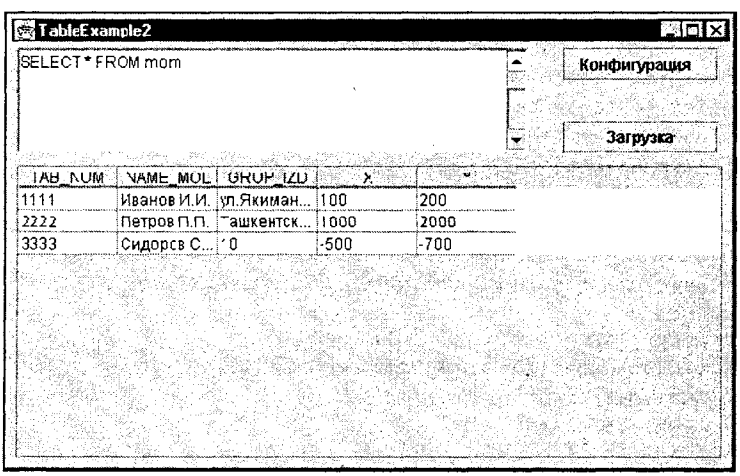


Рис. 6.9. Ввод запроса SQL и печать результатов

Если нажать кнопку мыши, например при выборе заголовка столбца GRUP_IZD, то записи в таблице будут отсортированы в соответствии с этим столбцом, как ключевым при сортировке (рис. 6.10).

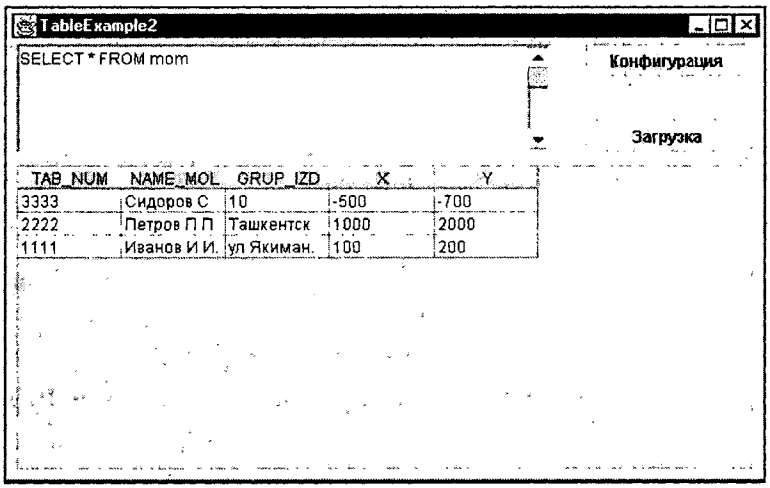


Рис. 6.10. Пересортировка записей в таблице по событию нажатия мыши

Все эти методы программирования на Java легко можно применить практически в любой программной среде, ориентированной на создание Web-приложений и распределенных Интернет-приложений. Разумеется, способы доступа к данным этим не исчерпываются. Многие разрабатывают очень эффективные Интернет-приложения с использованием Perl-скриптов и других скриптовых языков программирования. Крупные компании, подобные IBM и Oracle, разрабатывающие программное обеспечение, предлагают разнообразные коннекторы для связи с базами данных или свои собственные технологии, типа встроенного языка JavaSQL (имеются его различные диалекты для Oracle, Informix и DB2). Но главное — это, все-таки, знание основополагающих стандартов и открытых протоколов, принятых различными международными организациями, наиболее авторитетной среди которых по-прежнему является ISO (International Standard Organization).

Приложение к главе 6. Интернет-ресурсы и источники

- С. Дунаев "Доступ к базам данных из Java-программ и проблемы русификации" — статья написана автором настоящей книги в октябре 1997 года и опубликована в Интернете на множестве десятков популярных сайтов.

В списке ссылок и адресов, приведенном ниже, можно найти много других полезных материалов, касающихся программирования для Интернета и работы с базами данных (ссылки и адреса Web выделены полужирным начертанием).

- Журнал "СУБД" (Издательство "Открытые Системы")

  http://osp.asu.pstu.ac.ru/dbms/

- Сервер Информационных Технологий содержит море (!) аналитической информации

  http://www.citforum.ru/index.html

- Виртуальная энциклопедия "Linux по-русски": программирование под Linux

  http://linux-ve.chat.ru/

- Library of Athena

  http://athena.vvsu.ru/docs/

- Учебник

  http://programmerland.narod.ru/ushebnik.htm

- Web-мастеру

  http://webmaste.diaspora.ru/dc/js.html

- Java. Интернет-ресурсы

  http://www.nntu.sci-nnov.ru:8100/DISLRN/javalinx.htm

- Рэйд-Квадрат – Программирование

  http://www.raid-square.ru/docs/prog.html

- The Java Tutorial

  http://javatutorial.chat.ru/

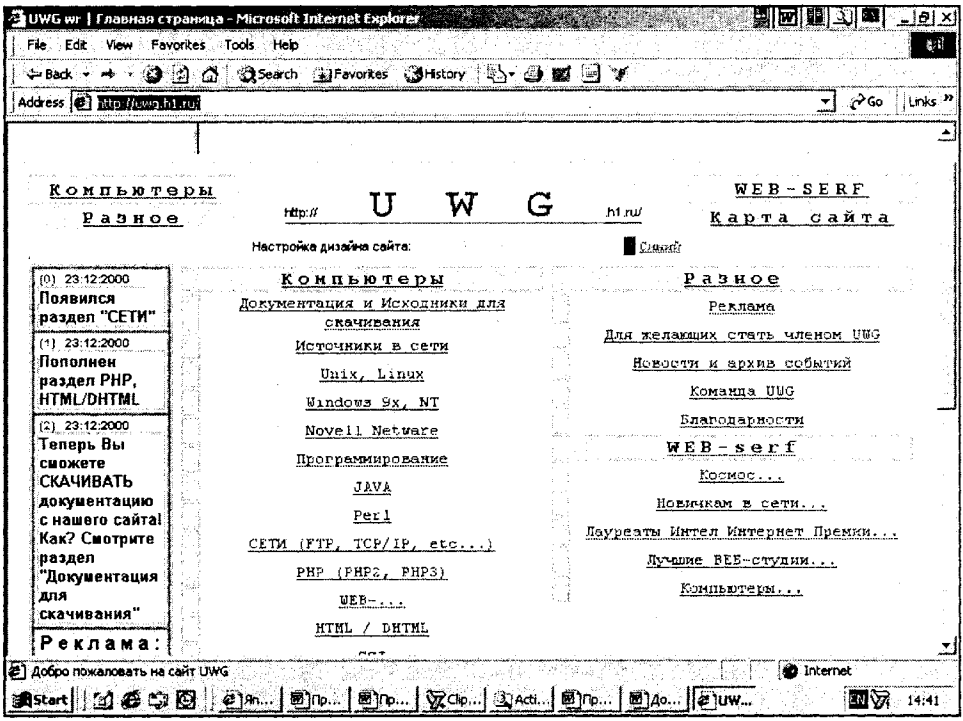
- Архив компьютерной документации

  http://zareg.newmail.ru/db/db07.htm

- Город-Инфо

  http://infocity.kiev.ua/

- Сайт UWG



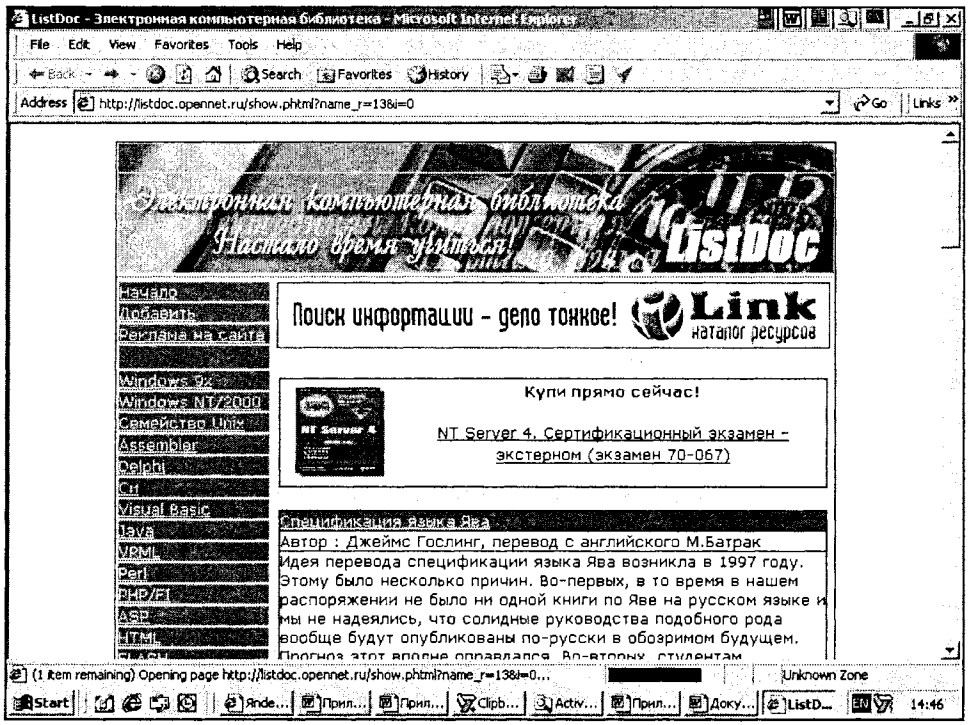
- Hardware, Software, Programming info collection
  http://classic.tep.ru/serge/computechinfo/index.htm
- JavaPower. О яве главная страница
  http://chat.ru/~k_v_v/javapower/java/main.htm
- Библиотека программиста — JAVA и JAVA Script >> Литература
  http://prog.agava.ru/JAVA/
- Компьютерная литература — документы online
  http://complit.timus.ru/complit_on.asp?id=java
- LinuxCenter — путеводитель в мире Unix-систем. Дистрибутивы RedHat, Mandrake
  http://www.linuxcenter.ru/enc/algol.phtml
- **Java/VB-script — (www.mstu.ru)** неофициальный сайт МГТУ им Н. Э. Баумана

- SOFT — Обзоры лучших программ — The OpenNet Project: Поиск ресурсов в каталогах ссылок и программ (OpenNET)

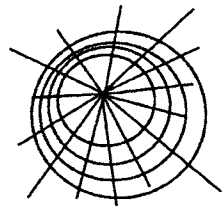
  http://www.iden.ru/cgi-bin/opennet/dir_search.cgi?mask=.*&last_day=0&type=2&system=0&dir=1&lines=0

- ListDoc — Электронная компьютерная библиотека

  http://listdoc.opennet.ru/show.phtml?name_r=13&i=0



ГЛАВА 7



Java и XML как новый способ работы с данными и СУБД

Зачем нужен XML

XML был вызван к жизни объективными обстоятельствами. Почти все фирмы стремятся перевести свои существующие приложения на Web-платформу. Казавшиеся некогда почти незаменимыми "настольные" приложения, постепенно вытесняются универсальным интерфейсом, встроенным в браузер. Вероятно, наступит время, когда кроме браузера в компьютере пользователя не останется почти ничего, что могло бы присутствовать на экране в качестве альтернативы данному универсальному интегратору всех программных ресурсов. В связи с этим возникло множество новых проблем, которые блестяще разрешились с возникновением нового стандарта. Явление XML удивительным образом напоминает массовый психоз восторженности, когда появился Java. Так же крупнейшие производители программного обеспечения заявили о своей поддержке XML, компьютерные журналы наводнились сотнями и тысячами статей об XML, появились глашатаи революционного обновления, призывающие, как всегда, сбросить прежних кумиров с корабля современности. Но если отбросить всю эту рекламную шумиху и поглядеть на все трезво, то нельзя искренне не восхититься удивительной простоте и изяществу нового и, как всегда, старого стандарта, который в полной мере отвечает вызову наступившего века.

Суть XML, если взять его основную идею, проста и красива. XML представляет собой тип базового синтаксиса, который может использоваться для любого числа документов или форматов данных. Он блестяще преодолевает недостатки HTML, в котором "смешались в кучу кони, люди", но идея маркировки текстов с помощью тегов, т. е. специальных маркировочных конструкций, остается в полной неприкосновенности. Просто в XML все четко определено. Имеется, в отличие от документа HTML, три файла, связанных с конкретным документом.

- Сами данные в их первозданной и наглядной структурной иерархии (файл с расширением .xml).
- Стилевое описание представления на экране или при печати каждого элемента данных (документ с расширением .xsl).
- Правила синтаксиса, согласно которым должен строиться документ (файл с расширением .dtd).

Благодаря этим железным правилам достигается необыкновенная свобода при описании любых типов данных и структур, возможности их изображения в произвольных, даже самых непредсказуемых форматах, возможность проектирования своего самого изощренного синтаксиса, согласно которому должен строиться документ. Если сам файл XML обязателен, то два оставшихся — вовсе нет, можно воспользоваться стандартными стилевыми и синтаксическими описаниями текстовых документов.

XML появился не с нуля. Так же как и HTML, он является подмножеством суперязыка разметки текстовых документов — легендарного SGML (ISO 8879. Information Processing-Text and Office Systems — Standard Generalized Markup Language (SGML), 1986). Однако HTML, с течением времени, порождал все больше и больше проблем и тихо стал раздражать разработчиков. Кому, скажем, может понравиться необходимость повторения фрагментов кода на каждой странице, например, логотипа фирмы или навигационного столбца с меню. Для решения этой проблемы стали использовать метод подстановок, как при программировании макроопределений. Так появились Server Site Includes (SSI, вставки на уровне сервера). Вставка внешнего файла в HTML-страницу — это самый простой способ применения SSI:

```
<html>
<head>
...
</head>
<body>
...
<!-#include virtual= "file.htm" ->
...
<!-#include file= "file.htm" ->
...
</body>
</html>
```

Однако, чтобы использовать этот метод, сервер обязан являться интерпретатором языка SSI, т. е. в составе модулей сервера должен быть и модуль интерпретации SSI.

Потом появились каскадные таблицы стилей (CCS, Cascading Style Sheets), патентом на спецификацию которых владеет Microsoft. Основная идея CSS состоит в том, чтобы отделить логическую структуру документа от формы

его представления. Тем не менее, как впоследствии оказалось, все это были промежуточные этапы.

С появлением в феврале 1998 г. спецификации метаязыка XML 1.0 и принятием его консорциумом открытых стандартов W3, а также поддержкой этого формата браузерами Microsoft Internet Explorer 4.01, 5.0, Web-серверами Apache, IIS и заявлением разработчиков браузера Netscape Navigator о поддержке XML стали началом новой эпохи в программировании для Web.

Предположим, что ваша компания производит и распределяет товары через сеть торговых посредников, которые в свою очередь продают товары конечным потребителям. Когда торговые посредники заказывают некоторые изделия, заказы посылаются через корпоративную сеть, с использованием электронного формата обмена данными (EDI). Ваша компания выставляет счета через тот же самый механизм, используя другие форматы EDI, те, которые разработаны для счетов. В общем виде процесс выглядит так:

1. Ваш торговый посредник вводит заказ через свою локальную систему.
2. Заказ оформляется в формате EDI и посылается в вашу систему обработки заказов. Для того чтобы это могло работать, торговый посредник должен быть клиентом корпоративной сети и использовать общие для этой сети интерфейсы.
3. Система обработки заказов планирует заказ к выполнению и перенаправляет его в систему обработки счетов (invoice system).
4. Система обработки счетов посылает клиенту счет с суммой, которую он должен оплатить также в специальном формате EDI (для него необходим дополнительный интерфейс).
5. Локальная платежная система торгового посредника обрабатывает полученный счет, производит оплату и перемещает оплаченный счет к системе обработки счетов.

Представленная система чрезвычайно громоздка, и поэтому сведения о новых торговых посредниках не могут быть быстро и легко добавлены к сети. Любое внесение изменений превращается в серьезную проблему при наличии множества разнообразных клиентов.

XML разработан, чтобы решить эту задачу. Теперь вместо того, чтобы покупать сложные системы, поддерживающие EDI, достаточно разработать документы в формате XML и использовать Интернет вместо дорогостоящих корпоративных сетей.

Суть XML

Структура любого XML-документа представляет собой иерархию тегов (с одним корневым), которая задает связи между ними, а сами теги определяют семантику документа. Например, опишем в XML-структуру некоего учебного заведения.

```
<Institute Name="Институт"> Государственный университет
  <Faculties Name="Факультеты">
    <Faculty Name="Факультет">Технический</Faculty>
    <Faculty Name="Факультет">Гуманитарный</Faculty>
  </Faculties>
  <Chairs Name="Кафедры">
    <Chair Name="Кафедра">Высшей математики</Chair>
    <Chair Name="Кафедра">Физики</Chair>
    <Chair Name="Кафедра">Истории</Chair>
    <Chair Name="Кафедра">Философии</Chair>
  </Chairs>
</Institute>
```

Для отображения XML-документов используется язык XSL (Extensible StyleSheet Language) совместно с JavaScript, Java или другими языковыми средствами, что позволяет создавать динамические страницы. XSL имеет встроенные возможности по извлечению информации из XML-документов, ее сортировке и фильтрации.

Для редактирования XML-документов существует множество XML-редакторов. В качестве примера можно привести XML-Notepad компании Microsoft, который отображает XML-документ в виде дерева с двумя колонками: в левой колонке показаны названия тегов и их атрибутов, в правой — их значения, что позволяет достаточно быстро вносить изменения в XML-документы.

Сильной стороной XML является его удивительная простота и наглядность. Не следует забывать, что XML-документ — это обычный текстовый документ, который может быть прочитан и через тысячу лет, как египетский папирус, чего нельзя сказать о специфической реляционной таблице (попробуйте найти устройство для чтения документов, созданных в бинарных форматах для компьютеров старших поколений). В отличие от них XML-документ может быть распечатан на бумаге и расшифрован, т. е. воспроизведен именно в том виде, как его задумал разработчик. И главное, что при этом, он может быть организован в сложную разветвленную древовидную структуру, которую даже физически легко обозреть. Представьте, что на огромном широком полу в закрытом спортивном зале вы выкладываете пачки документов, выстраивая их в некотором определенном порядке — так обычно люди сортируют свои архивы — пачка к пачке: письма, фотографии, документы, разложенные по годам, по принадлежности, по авторству и связанные с ними книги, значки, атрибутика. Все это, в конечном счете, образует древовидную причудливую структуру, подчиняющуюся четким и понятным создателю законам, которая, тем не менее, может быть легко описана, так, чтобы стать понятной и другим, легкой для ориентации в этом архиве и для нахождения требуемой записи или группы записей, связанной с тем или иным документом. Это и есть, по большому счету, XML-структура. Она предельно ясна, потому что в самой себе содержит собственные

описания. Эти описания легко читаются, и по ним можно создавать специальные инструменты для поиска всего самого нужного. К таким инструментам относятся *парсеры* и различные *объектные модели представления XML-структур*.

Парсеры нужны для получения доступа к структуре и содержимому XML-документа через программные интерфейсы, обычно из Java-программ (но не обязательно) в соответствии с некоторой объектной моделью XML-документа. В данной книге мы не будем подробно излагать синтаксис XML и приемы построения XML-документов, поскольку на эту тему уже написаны десятки книг и сотни статей. Куда интереснее взглянуть на проблему с другой стороны — какие выгоды можно извлечь из симбиоза четырех ведущих на сегодняшний день технологий: Java, СУБД, XML, Web? Поэтому, прежде чем приступить к описанию парсеров, определимся с терминами и понятиями, существующими в этой стремительно расширяющейся сфере.

Многоуровневые приложения на основе Java и XML

Сегодня существуют Web-серверы, Servlet-машины, реляционные и объектные базы данных. Имеется большая степень вероятности, что проектировщики на XML захотят использовать все эти прекрасно разработанные и освоенные технологии. Но как? Для того чтобы комбинировать средства, необходимо знать и о многом другом — какие существуют API, например, SAX, DOM, Servlets, Swing, RMI, JDBC и другие ключевые Java API. Если распределить все по уровням, то получится примерно такая картина:

1. Уровень приложений, написанных на любом объектно-ориентированном языке программирования (например, Java).
2. Уровень интерпретации содержимого XML-объектов (XML-Parser).
3. Уровень информационного хранилища, так называемые *persistence engines* (объектные базы данных, реляционные базы данных, файловые системы, Web-сайты). Информацию, находящуюся в этих универсальных хранилищах, легко преобразовать в XML и наоборот, используя объектную модель, построенную на основе Java-классов, получившую название DOM (Document Object Model).

Итак, на первом уровне профессионалами-программистами создаются программные модули, полностью абстрагированные от всякой конкретики, связанной с данными и с их представлениями, т. е. внешними интерфейсами. Все это выносится на уровень универсального описания метаданных на универсальном метаязыке — XML. (К настоящему времени еще недостаточно проработаны вопросы, связанные с генерацией самих описаний, акцент пока смещен в сторону интерпретации уже готового XML-содержания.) На этом уровне также могут существовать две стратегии.

Клиентские приложения

На рис. 7.1 изображен простейший тип приложения, когда информация сохраняется в XML-документах (файлах). В этом случае вы создаете свой собственный формат описания и представления данных в XML, который ничем не будет хуже того, в котором данные были бы организованы по иному принципу (например, в объектной или реляционной базе) — все операции над данными, включая их поиск и различные выборки по заданным критериям, также окажутся доступными через объектную модель XML. Классический случай, приводимый во множестве руководств по XML, — это адресная книга, которая может храниться не в СУБД, а непосредственно в XML-файле (рис. 7.2).

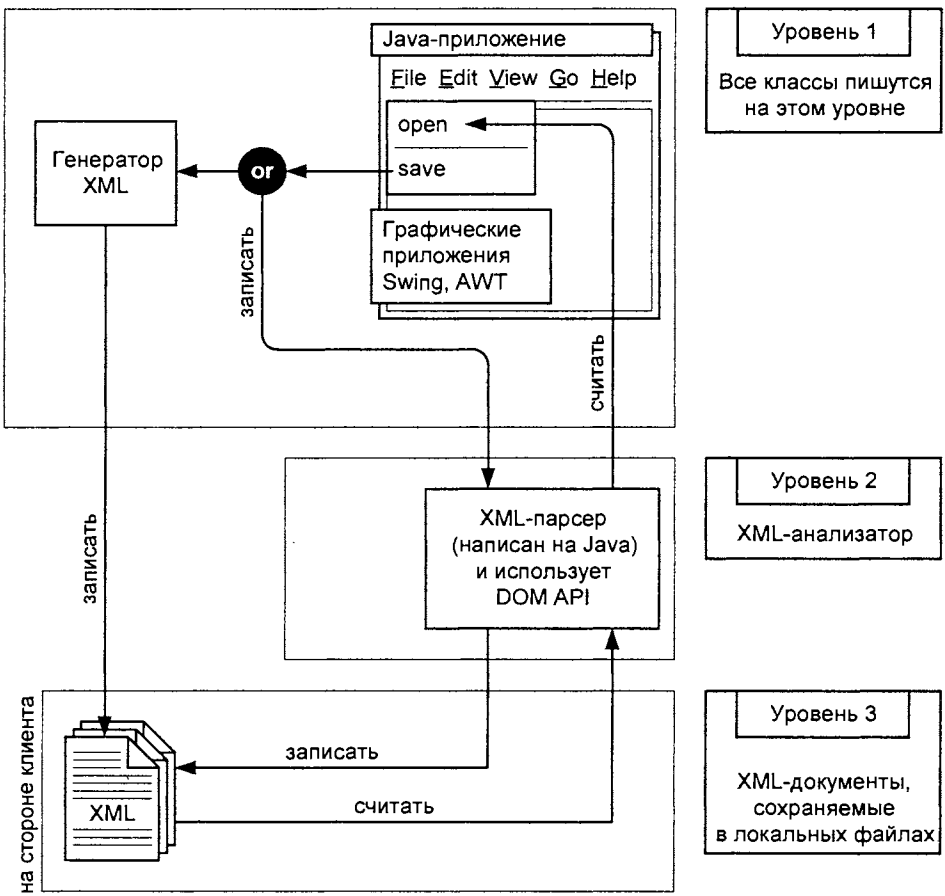
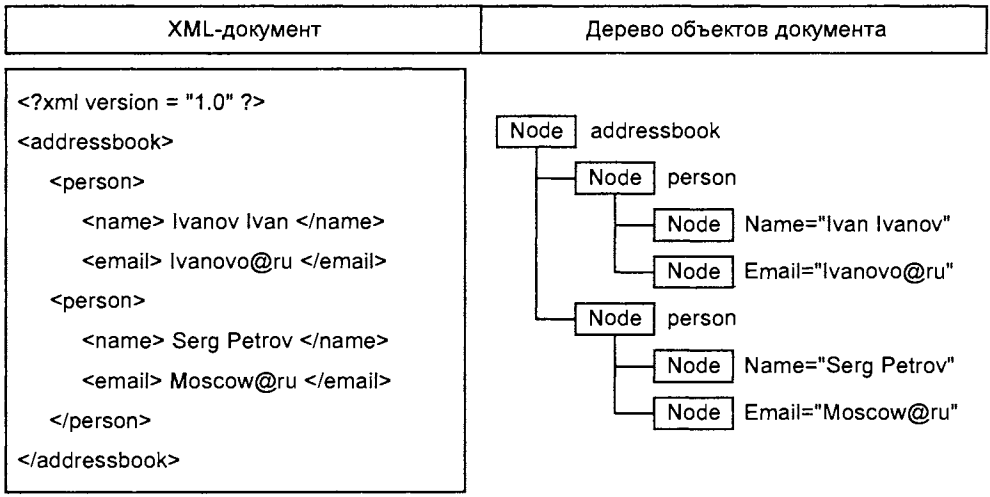


Рис. 7.1. Java-приложение со встроенной графикой



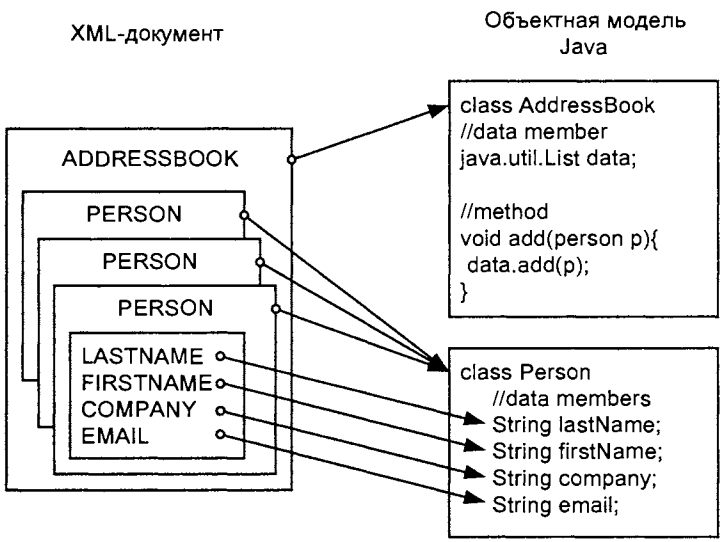


Рис. 7.2. Иерархическая структура объекта документа

Серверные приложения

В отличие от простейшего клиентского приложения, представленного на рис. 7.1, где Java-программа отображала данные из XML-файла, используя собственные графические средства и объектную XML-модель данных, сервер-ориентированные программы выполняются на сервере. На стороне клиента может быть либо обыкновенный браузер, либо миниатюрное приложение, обменивающееся по сети только XML-описаниями, существующими

в виде объектов. Сами данные и связанные с ними представления в виде списков, структур, иерархий, а также способов их отображения, могут быть получены по сети из самых разнообразных источников (рис. 7.3). Поскольку XML-документы позволяют хранить в себе или описывать структурированные данные, то любую информацию из СУБД можно представить в виде XML-файла. Именно этой цели и служит DOM. Рис. 7.4 иллюстрирует процесс прохождения и преобразования объектной информации. DOM предоставляет доступ к информации, сохраненной в документе XML в виде иерархической объектной модели, которая создает дерево узлов, отражающее структуру информации в вашем XML-документе, и программист может получать доступ к информации, взаимодействуя с этим деревом узлов.

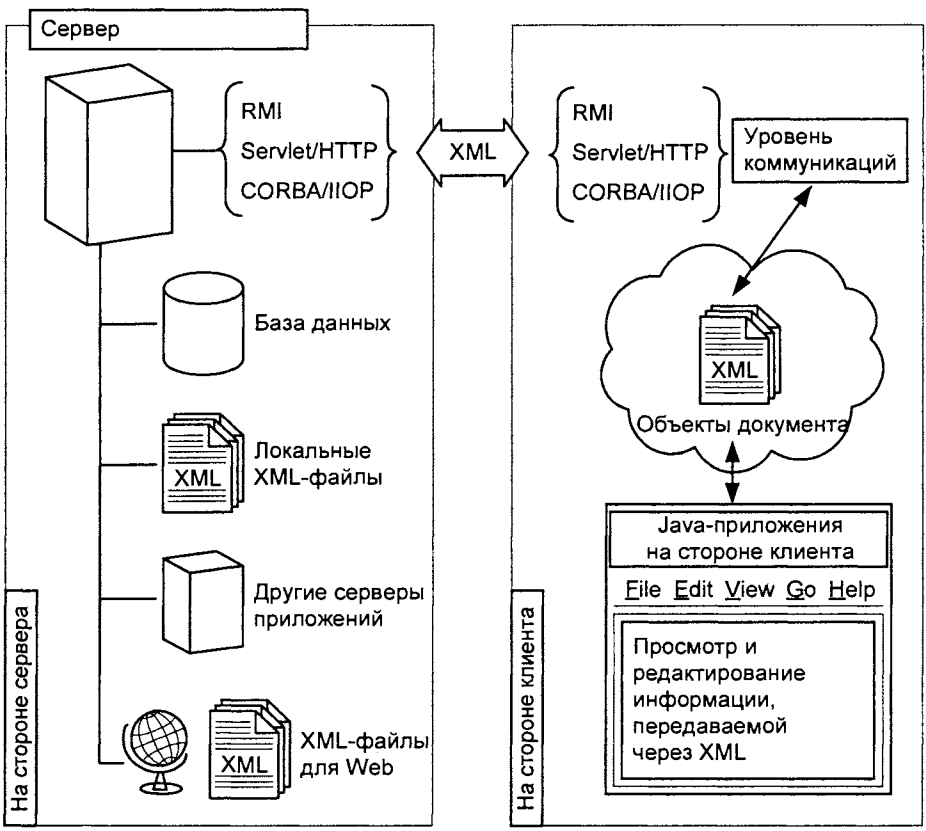


Рис. 7.3. Сервер приложений XML

Как все это можно применить на практике? Естественно, что лучше всего Java/XML-технология реализуется в многозвенных системах, где в качестве звеньев цепи выступают нижеприведенные субъекты.

- **Ультра-тонкий клиент**, иногда еще называемый XML-агентом, в функции которого входит разбор XML-документа, подключение таблицы стилей и соответствующее представление содержимого документа, как его изначально представлял себе разработчик. Некоторые XML-агенты преобразуют XML-документ к HTML-представлению. Однако Microsoft Internet Explorer 5, например, является XML-агентом, который не нуждается в таком преобразовании. Хотя, на сегодняшний день имеется весьма мало XML-агентов, предполагается, что число их будет постоянно увеличиваться вместе с ростом популярности XML.

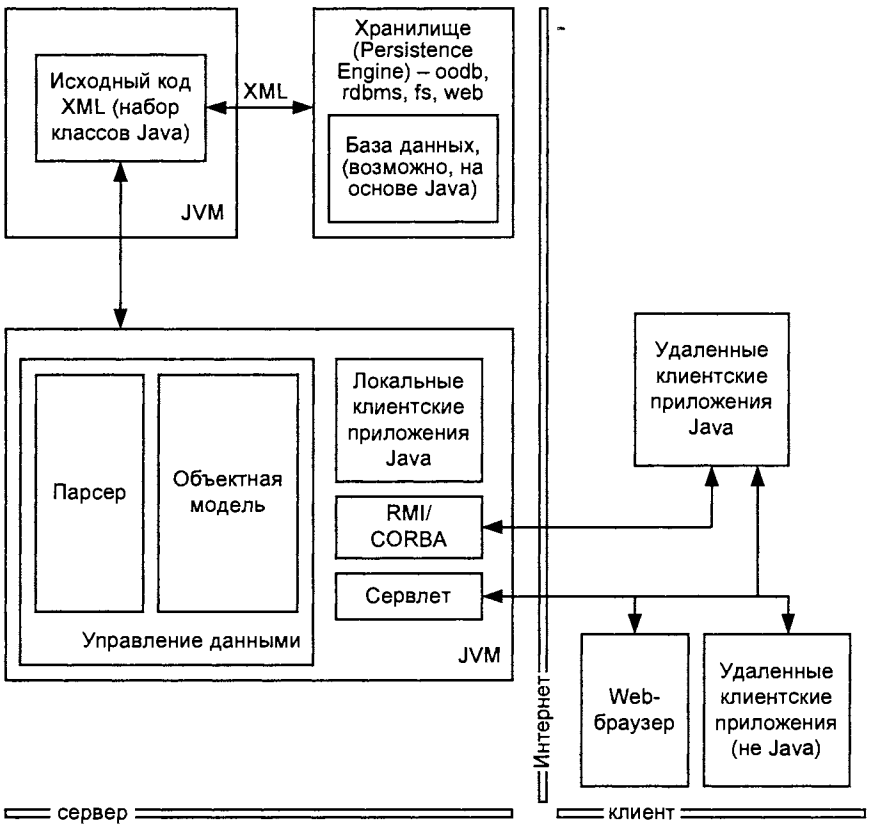


Рис. 7.4. Процесс прохождения и преобразования объектной информации

- **Сервер приложений** с возможностью интеграции с наиболее популярными Web-серверами и поддержкой многообразных сервисов, обслуживающих XML-структуру, таких как сервлеты, beans, EJB, JSP. Конечно, можно настроить Web-сервер, чтобы он обслуживал только статичные XML-документы, однако для обработки и динамической генерации XML-до-

кументов необходимо, чтобы он умел работать и с Java Server Pages, и с Enterprise JavaBeans и использовал сервлеты. Обычно такие расширения реализуются в виде набора дополнительных модулей и библиотек (plug ins).

- **Сервер базы данных**. Содержимое динамически генерируемых XML-документов может быть извлечено с серверов СУБД, таких как Oracle, Sybase, Informix, DB2. В зависимости от типа XML-приложения извлекаемые данные могут быть возвращены либо в формате XML-документа, либо в формате JDBC, который сервлет преобразует в структуру XML-документа. В будущем, вероятно, появится достаточное количество коммерчески доступных JavaBeans, которые будут содержать в себе удобную XML- и СУБД-специфику.

Типичная трехзвенная схема работы многочисленных подключенных клиентов через сервер приложений изображена на рис. 7.5.

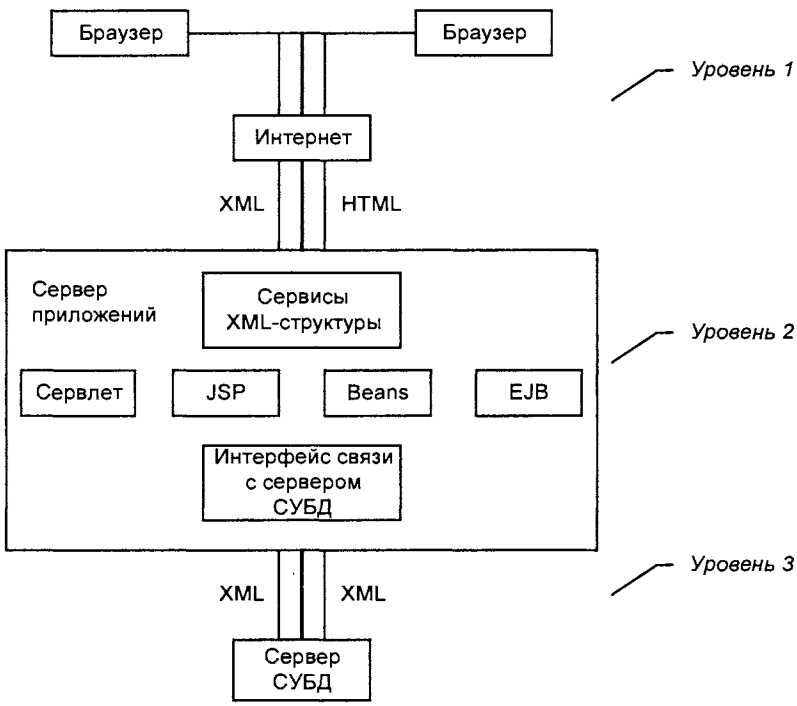


Рис. 7.5. Использование сервера

Охарактеризуем вкратце некоторые сервисы по обслуживанию структуры XML-документов.

- Сервлеты — специальные серверные компоненты, написанные на Java с использованием интерфейса JSDK и обменивающиеся информацией

с запустившим их браузером по протоколу HTTP. Сервлеты, в отличие от апплетов, гораздо функциональнее и быстрее, потому что выполняются на сервере и используют более мощную JVM (Java Virtual Machine, виртуальную машину Java).

- Beans[1] и EJB (Enterprise JavaBeans) — это повторно используемые компоненты, написанные на Java, которые могут встраиваться в любые модули и программные системы, включая графические средства программирования и серверы СУБД.

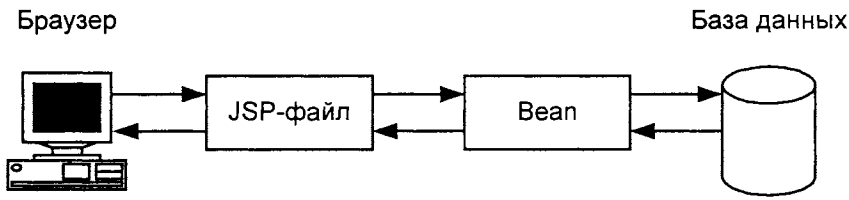


Рис. 7.6. Использование JSP

- Сервер приложений может и должен поддерживать новое мощное средство для изготовления динамических Web-страниц — JSP (Java Server Pages). JSP — это встроенное средство, базирующееся на Sun JSP-спецификации (рис. 7.6).

  Когда Web-сервер получает от браузера запрос на исполнение JSP-страницы (JSP-файл), он посылает запрос к серверу приложений. Сервер приложений разбирает JSP и генерирует Java-код, который выполняется как сервлет. Компиляция осуществляется только один раз, при первом вызове. JSP почти ничем не отличается от обычного HTML-файла и может содержать следующие теги:

  - `<servlet>` — для встраивания сервлетов в JSP;
  - директивы `<%...%>` или тег `<SCRIPT>` — для исполнения сценариев на JScript и VBScript;
  - Inline Java code — так называемые скриптлеты, т. е. встроенные в тело документа Java-операторы;
  - `<BEAN>` — тег для использования JavaBean-компонентов, который может задаваться в форме:

```
<bean name="Bean_name" varname="local_Bean_name"
type ="class_or_interface_name" introspect="yes|no"
```

[1] В переводе означает "зерна кофе".

```
beanName="ser_filename" create="yes|no"
scope="request|session|userprofile" >
<param property_name="value">
</bean>
```

Например:

```
<!-- Объявление Bean -->
<bean name="foobar" type="FooClass" scope="request" >
<!- Определение некоторых свойств -->
<param fooProperty="fooValue" barProperty="1">
</bean>
<!- Впоследствии можно использовать некоторые методы этого Bean -->
<p>Имя колонки - <%= foobar.getRowName() %>.</p>
```

Таким образом, функциональность Web-серверов расширяется до того, что используются не только автоматически генерируемые HTML-страницы, но и встроенные в них средства связи и извлечения данных из СУБД и XML-структур. Компоненты JavaBean, которые выполняют черновую работу, могут быть файлами типа class, компонентами, подготовленными для транспортировки по сети (serialized Beans), или динамически генерируемыми сервлетами.

Обобщая сказанное, можно утверждать, что XML в настоящее время является идеальным способом описания и хранения любых данных, в том числе и структурированных, извлеченных из реляционных баз данных. XML использует объектную модель и предоставляет объектные интерфейсы для доступа к описанным или содержащимся в нем данным, которые с успехом можно применять из любого объектно-ориентированного языка программирования, но лучше всего к этому приспособлен Java.

Какие бывают парсеры

Прежде чем говорить о парсерах, скажем сразу, что типичным парсером является обычный браузер. Именно он разбирает по косточкам документ в формате HTML, со всеми встроенными в него программными "штучками" в виде Java-апплетов, скриптов, ActiveX-элементов, чтобы все правильно отобразить на экране. XSL-процессоры, например Lotus XSL Processor, которые разбирают структуру XML-документа на предмет его стилевого оформления в соответствии с приставленным к этому документу файлом с расширением .xsl — тоже парсеры, ведь стилевое описание документа XML на языке XSL может быть весьма сложным. XSL содержит в себе элементы языка программирования, такие как `xsl:process select`, `xsl:if` и `xsl:choose`. В итоге работы парсера, разбирающего XML-документ, чаще всего генерируется обычный HTML-документ, который передается в браузер. Но все дело в том, что формат HTML в качестве основного результата вовсе не является

единственным. Главное — это понять структуру и способ представления данных в документе, которые описаны на метаязыке XML! Что делать потом с этими данными — решит пользовательское приложение. Прелесть XML заключается в том, что любую, самую сложную структуру документа, описанного на этом языке, можно представить в виде объектной модели, с которой удобно и легко работать в объектно-ориентированных системах программирования, например, в Java.

Парсер является важной частью связки Java и XML, но в то же время, это всего лишь небольшая часть целой картины. Программист получает доступ к XML-документам (обычно сохраняемым на сервере) посредством сервисов парсера Java XML. Имеется два вида парсеров — SAX- и DOM-парсеры. Обе разновидности SAX и DOM API позволяют программистам получать доступ к информации в XML-документах.

Интерфейсы SAX и DOM API являются достаточно сложными. SAX лучше подходит для чтения XML-содержания из документов, которые генерируются другим программным обеспечением, а DOM — для считывания информации из сохраненных документов. Так, если ваш XML-документ содержит машинно-генерируемые данные, то лучше всего применить SAX. Если это обычный XML-файл — можно использовать DOM. В принципе, никаких ограничений не существует, и это всего лишь рекомендация.

Синтаксический анализатор (парсер) дает возможность обратиться к данным посредством XML-документов непосредственно из Java-программ. Как только данные были прочитаны, они сохраняются в виде некоей объектной структуры, над которой можно производить требуемые манипуляции, используя объектные спецификации.

SAX не обеспечивает объектную модель по умолчанию. Когда применяется SAX, вы получаете возможность создавать свою собственную объектную модель Java для представления этих данных, используя Java-объекты. С другой стороны, DOM обеспечивает объектную модель по умолчанию. Согласно этой модели все элементы XML-документа трактуются как дерево, состоящее из Java-объектов.

Другая важная вещь, которую вы должны решить для себя при использовании парсеров, — это возможность написания кода, который позволял бы сохранять ваши собственные объектные модели в виде XML-файлов. Это очень важное обстоятельство, учитывая универсальность XML.

Суть DOM

DOM дает возможность получать доступ к информации, сохраненной в XML-документе, как к иерархической объектной модели. DOM создает дерево узлов (основанное на структуре XML-документа), и вы можете получать доступ к информации, взаимодействуя с этим деревом. Текстовая ин-

формация в вашем XML-документе превращается в связку структурно организованных узлов (см. рис. 7.2).

Рис. 7.2 несколько упрощен, потому что в DOM каждый узел элемента фактически включает список других узлов как дочерние записи. Эти узлы дочерних записей могли бы содержать текстовые значения или являться другими узлами.

Суть SAX

SAX обеспечивает доступ к информации в вашем XML-документе, рассматривая его не как дерево узлов, а как последовательность событий. Вы можете спросить, а чем это будет полезно? Ответ прост — поскольку SAX не создает объектной модели, начиная с вершины XML-документа (подобно DOM), обработка идет быстрее, однако, при этом вы сами должны позаботиться о создании:

- вашей собственной объектной модели;
- класса, который обрабатывает SAX-события и в соответствии с ними строит объектную модель.

При использовании DOM в этих действиях нет необходимости.

В случае DOM парсер делает почти все: читает XML-документ, создает объектную модель и затем строит ссылки по каждому элементу этой модели, чтобы вы могли ею управлять. SAX не зря называется простым интерфейсом (Simple API for XML), потому что он действительно прост. Единственное чего SAX требует, так это того, чтобы парсер читал XML-документ и обрабатывал события, связанные с чтением определенных элементов этого документа. Ответственность за интерпретацию событий возлагается не на парсер, а на приложение, которое использует этот интерфейс.

Какие виды SAX-событий обрабатываются парсером? SAX порождает событие при каждом встреченном открытом теге и, соответственно, закрытом. Кроме того, генерируются события при обработке секций PCDATA и CDATA. Обработчик событий при чтении документа должен интерпретировать эти события некоторым образом и создавать вашу собственную объектную модель, основанную на них.

Когда использовать DOM

Если документ имеет простую и ясную схему, DOM является вполне естественным выбором. Наиболее приемлемыми вариантами DOM представляются хорошо организованные списки документов, собранные в некоторую структурную иерархию, подобно справочникам и каталогам. Однако если вы имеете дело со сложными и на первый взгляд непонятно каким образом

структурированными данными со множеством обратных взаимосвязей, то в таком случае, вероятно, предпочтительнее использовать SAX.

Когда использовать SAX

Если информация, сохраненная в ваших XML-документах машиночитаема (и сгенерирована программным обеспечением), то лучше всего обеспечить доступ к такой информации при помощи SAX. Машиночитаемые и сгенерированные данные обычно включают в себя следующие вещи:

- свойства (properties) Java-объектов, сохраненные в XML-формате;
- запросы, оформленные на языках типа SQL, XQL, OQL;
- результирующие наборы данных, сгенерированные этими запросами (они могут включать данные в виде реляционных таблиц, декодированных в XML).

Пример DOM

Приведенный ниже пример демонстрирует способ конструирования полноценного XML-документа при помощи Java-программы, использующей SUN Java XML-парсер, в данном случае работающий по модели DOM. Парсер существует в виде файла xml.jar, который свободно доступен на официальном сайте **http://java.sun.com**.

```
import java.io.*;
import com.sun.xml.tree.*;
import org.w3c.dom.*;

public class main
{
  //
  // Конструирование XML-документа
  //
  public static void main (String argv [])
    throws IOException, DOMException
    {
      XmlDocument  doc = new XmlDocument ();
      ElementNode  root = (ElementNode) doc.createElement ("root");
      Attr         tmp;
      Writer       out = new OutputStreamWriter (System.out);

      doc.appendChild (root);
      out.write ("No children ...\n");
      doClones (out, root);
      out.write ("\n");
```

```
    root.appendChild (doc.createElement ("header"));
    root.appendChild (doc.createTextNode ("\n  some data is text\n  "));
    root.appendChild (doc.createElement ("footer"));

    // Пробелы добавляются, чтобы удобнее было читать XML-документы

    Text  text;

    // Переносимый метод получения первого элемента из корня — это
    // root.getChildren ().item (0) ... требуется также проверка,
    // не пуст ли список потомков (дочерних элементов), поэтому при
    // безопасном кодировании этот код займет не одну, а несколько строк.
    text = doc.createTextNode ("\n");
    root.insertBefore (text, root.item (0));
    text = doc.createTextNode ("\n");
    root.appendChild (text);
    out.write ("... mixed content model ...\n");
    doClones (out, root);
    out.write ("\n");

    // Здесь показан один из способов создания атрибута...
    tmp = doc.createAttribute ("attr1");
    tmp.setNodeValue ("value of attribute 1");
    root.getAttributes ().setNamedItem (tmp);

    // ... и еще один (проще и более предпочтительный)
    root.setAttribute ("attr2", "another string");
    out.write ("... and attributes...\n");
    doClones (out, root);
    out.write ("\n");

    // для полноты счастья можно добавить и DTD
    doc.setDoctype (
    null,     // no public identifier
    "http://www.example.com/xml/dtd1.dtd",
    null      // no internal subset
   );

    out.write ("DOCUMENT is:\n");
    doc.write (out);
    out.write ("\n");
    out.flush ();

    System.exit (0);
  }

 private static void doClones (Writer out, ElementNode root)
   throws IOException
```

```
    {
      out.write ("ORIGINAL:\t");
      root.write (out);
      out.write ("\n");
      out.flush ();
      {
        ElementNode  clone1;
        clone1 = (ElementNode) root.cloneNode (false);
        out.write ("SHALLOW CLONE:\t");
        clone1.write (out);
        out.write ("\n");
        out.flush ();
      }

     {
        ElementNode  clone2;
        clone2 = (ElementNode) root.cloneNode (true);
        out.write ("DEEP CLONE:\t");
        clone2.write (out);
        out.write ("\n");
        out.flush ();
      }
    }
}
```

Компиляция программы

```
javac -classpath ..\..\xml.jar;%JAVA_HOME%\lib\classes.zip main.java
```

Выполнение программы

```
java -classpath .;..\..\xml.jar;%JAVA_HOME%\lib\classes.zip main
```

Результат выполнения программы

Трассировка выполнения и печать сгенерированного XML-файла:

```
Без дочерних ответвлений ...
ORIGINAL:       <root />
SHALLOW CLONE:  <root />
DEEP CLONE:     <root />

... смешанная модель ...
ORIGINAL:       <root>
<header />
```

```
  некоторые текстовые данные
  <footer />
</root>
SHALLOW CLONE:  <root />
DEEP CLONE:     <root>
<header />
  некоторые текстовые данные
  <footer />
</root>

... и атрибуты ...
ORIGINAL:       <root attr1="value of attribute 1" attr2="another
string">
<header />
  некоторые текстовые данные
  <footer />
</root>
SHALLOW CLONE:  <root attr1="value of attribute 1" attr2="another string" />
DEEP CLONE:     <root attr1="value of attribute 1" attr2="another
string">
<header />
  некоторые текстовые данные
  <footer />
</root>
```

Результирующий документ:

```
<?xml version="1.0" encoding="Cp1251"?>

<!DOCTYPE root SYSTEM 'http://www.example.com/xml/dtd1.dtd'>

<root attr1="value of attribute 1" attr2="another string">

  <header />
  некоторые текстовые данные

  <footer />

</root>
```

Пример SAX

Текст программы (http://java.sun.com/xml)

```
import java.io.File;
import com.sun.xml.parser.Resolver;
import com.sun.xml.tree.XmlDocument;
```

```
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

public class main
{
  //
  // Чтение и вывод содержимого XML-документа,
  // сохраненного в некотором файле.
  //
  public static void main (String argv [])
  {
    InputSource  input;
    XmlDocument  doc;

    if (argv.length != 1) {
      System.err.println ("Usage: cmd filename");
      System.exit (1);
    }

    try {
      // turn the filename into an input source
      input = Resolver.createInputSource (new File (argv [0]));
      // turn it into an in-memory object
      // ... the "false" flag says not to validate
      doc = XmlDocument.createXmlDocument (input, false);
      // normalize text representation
      doc.getDocumentElement ().normalize ();
      // prettyprint
      doc.write (System.out);
    } catch (SAXParseException err) {
      System.out.println ("** Parsing error" +
                          ", line " + err.getLineNumber () +
                          ", uri " + err.getSystemId ());
      System.out.println("   " + err.getMessage ());
      // print stack trace as below
    } catch (SAXException e) {
       Exception  x = e.getException ();
       ((x == null) ? e : x).printStackTrace ();
    } catch (Throwable t) {
       t.printStackTrace ();
    }

    System.exit (0);
  }
}
```

Исходный файл book-order.xml

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<!DOCTYPE Order [

<!--
  В этом файле описываются синтаксические правила формирования тегов
документа, что вполне естественно для системы онлайновых интерактивных
систем обработки заказов и взаимодействия торговых партнеров.
-->

  <!ELEMENT Order (Customer,Manifest,Receipt)>
  <!ATTLIST Order xmlns CDATA #FIXED
"http://www.example.com/myschema.xml">

  <!ELEMENT Customer (Name, Cardnum)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Cardnum (#PCDATA)>

   <!ELEMENT Manifest (Item*)>

  <!ELEMENT Item (ID,Title,Quantity,UnitPrice)>
  <!ELEMENT ID (#PCDATA)>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Quantity (#PCDATA)>
  <!ELEMENT UnitPrice (#PCDATA)>
  <!ELEMENT Receipt (Subtotal,Tax,Total)>
  <!ELEMENT Subtotal (#PCDATA)>
  <!ELEMENT Tax (#PCDATA)>
  <!ELEMENT Total (#PCDATA)>

]>

<Order>
  <Customer>
    <Name>Bill Buckram</Name>
    <Cardnum>234 234 234 234</Cardnum>
  </Customer>
  <Manifest>
    <Item>
      <ID>209</ID>
      <Title>
        Duke: A Biography of the Java Evangelist
      </Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
```

```
    <Item>
      <ID>208</ID>
      <Title>
        100% Pure: Making Cross Platform Deployment a Reality
      </Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
    <Item>
      <ID>204</ID>
      <Title>
        Making the Transition from C++ to the Java(tm) Language
      </Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
    <Item>
      <ID>202</ID>
      <Title>Web Servers for Fun and Profit</Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
    <Item>
      <ID>210</ID>
      <Title>
        I Think Not: Dukes Likeness to the Federation Insignia
      </Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
  </Manifest>
  <Receipt>
    <Subtotal>$53.75</Subtotal>
    <Tax>$4.43</Tax>
    <Total>$58.18</Total>
  </Receipt>
</Order>
```

Компиляция программы

```
D:\XML\xml-tr2\examples\simple>javac -classpath
..\..\xml.jar;d:\jdk1.2\lib\classes.zip main.java
```

Выполнение программы

```
D:\XML\xml-tr2\examples\simple>java -classpath
.;..\..\xml.jar;d:\jdk1.2\lib\classes.zip main ..\samples\book-order.xml
```

О DOM и SAX API

XML API обычно существуют в двух видах: основанные на *событиях* и на *использовании деревьев*. Первый тип API (подобный SAX) использует асинхронные вызовы (callback), чтобы сообщать о событиях синтаксического анализа приложению. Приложение имеет дело с этими событиями через специально настроенные драйверы обработки событий. События включают начало и конец элементов и строк. В отличие от использования деревьев, событийно-основанные API обычно не выстраивают в памяти древовидную структуру XML-документа. Следовательно, SAX полезен для прикладных программ, которые не должны управлять XML-деревом, для операций типа поиска и других.

API, основанный на деревьях (такой, как DOM) строит в памяти дерево представления XML-документа. К тому же он обеспечивает приложение классами и методами для навигации по этому дереву. Обычно DOM интерфейс полезнее структурных манипуляций, таких как извлечение элементов в определенном порядке, добавление, удаление элементов, их переименование и т. д.

Использование XML в Oracle

Разумеется, что современный Oracle имеет полную гамму продуктов для поддержки XML. Все они:

- Oracle8*i* и *inter*Media
- XML Parsers и XSL Processors (Java, C, C++, and PL/SQL)
- XML Class Generators (Java and C++)
- XML SQL Utility for Java
- XSQL Servlet
- XML Transviewer Beans

входят в специальный пакет Oracle XML Developer's Kit (XDK), который, в свою очередь, является составообразующим компонентом Oracle8*i*.

Oracle8*i* и *inter*Media

Oracle8*i* включает собственную поддержку Интернет-стандартов, в том числе Java и XML, благодаря встроенной виртуальной Java-машине. Вы можете использовать *inter*Media Text, чтобы осуществлять поиск в XML-документах, сохраненных в Oracle8*i*, потому что они индексируются как обычные текстовые документы.

Когда вы сохраняете XML-документы в Oracle8*i*, имеются три базовые стратегии:

- сохранение XML-документа как обычного CLOB- или BLOB-объекта;
- сохранение XML-документа в качестве данных в объектно-реляционных таблицах;
- комбинирование XML-документов и данных, используя специальные виды (views).

Сохранение XML-документов как обычных документов

Сохранение XML-документов в виде объектов CLOB или BLOB является обычной стратегией, когда надо передавать пользователю полное содержание, подробно описываемое в XML-документе, и база данных здесь используется только как идеальное место хранения таких документов, их идентификации, извлечения и передачи по месту назначения. Таким образом могут сохраняться книги, брошюры, рекламные материалы и т. д.

Сохранение XML-документов в качестве данных

Если XML-документы имеют сложную структуру и содержат данные, которые постоянно обновляются, их лучше сохранять в качестве данных. Обычно XML-документы содержат элементы или атрибуты, которые могут быть представлены в виде объектно-реляционных расширений, таких как объектные типы, объектные ссылки и наборы структур. Oracle предоставляет богатые средства манипулирования подобными документами и элементами их структур. При этом может использоваться язык SQL и утилиты XML SQL.

XML-парсеры в Oracle

Oracle обеспечивает набор XML-парсеров для Java, C, C++ и PL/SQL (рис. 7.7). Каждый из этих парсеров является стандартным XML-компонентом, который разбирает конструкции XML и интерпретирует их в языке программирования. Парсеры поддерживают интерфейсы DOM (Document Object Model) и SAX (Simple API for XML), а также XML-пространство имен и XSL-преобразования. Одни и те же парсеры доступны на всех платформах Oracle.

Поддержка XSL-преобразований

XML-парсеры в Oracle тесно интегрированы с XSL-процессором (XSLT) для интерпретации способа представления данных, содержащихся в XML-файле.

Используя XSLT-процессор, можно преобразовывать XML-документы в HTML или другие форматы (рис. 7.8).

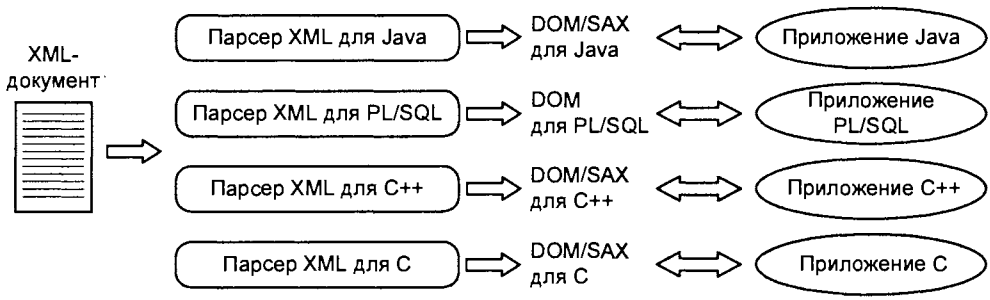


Рис. 7.7. Набор парсеров в Oracle

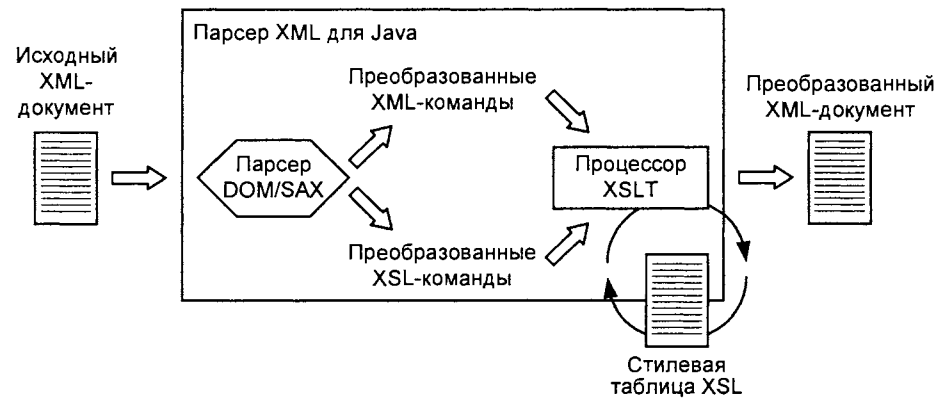


Рис. 7.8. Использование XSL в Oracle

Поддержка Namespace

Парсеры Java, C и C++ также поддерживают XML Namespaces. Пространство имен (Namespace) — это механизм разрешения коллизий при именовании тегов, типовых элементов и атрибутов в XML-документах, который обеспечивает универсальный механизм именования. К примеру, при использовании Namespace всегда можно отличить элемент данных с именем <EMP> в Oracle от другого элемента с таким же именем.

Пример Java XML-парсера

Следующий код Java содержит XML Parser for Java v2 для анализа содержимого XML-документа, затем использует DOM API для манипуляции XML-

данными и, наконец, для представления данных с помощью XSLT Processor API для преобразования их по собственным стилевым описаниям.

```
import org.w3c.dom.*;

import java.util.*;
import java.io.*;
import java.net.*;

import oracle.xml.parser.v2.*;

public class XSLTransform
{
  public static void main (String args[]) throws Exception
  {
    DOMParser parser;

    XMLDocument xml, xsldoc, out;

    URL xslURL;
    URL xmlURL;

    try
    {
      if (args.length != 2)
      {
        // Передать имена XSL и XML-файлов
        System.err.println("Usage: java XSLTransform xslfile xmlfile");
        System.exit(1);
      }

      // Осуществить разбор XSL и XML-документов
      parser = new DOMParser();
      parser.setPreserveWhitespace(true);

      xslURL = createURL(args[0]);
      parser.parse(xslURL);
      xsldoc = parser.getDocument();

      xmlURL = createURL(args[1]);
      parser.parse(xmlURL);
      xml = parser.getDocument();

      // Создать экземпляр таблицы стилей
      XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);

      XSLProcessor processor = new XSLProcessor();

      // Отобразить сообщения и предупреждения
      processor.showWarnings(true);
      processor.setErrorStream(System.err);
```

```
        // Обработка XSL
        DocumentFragment result = processor.processXSL(xsl, xml);

        // Создание выводного документа
        out = new XMLDocument();

        // Создать объект-заготовку для выводного документа
        Element root = out.createElement("root");
        out.appendChild(root);

        // Добавить измененное дерево к элементу документа-заготовки
        root.appendChild(result);

        // Печатать измененный документ
        out.print(System.out);
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
    }
}
```

XML-генератор классов для Java

XML Class Generator for Java создает исходные файлы Java из XML DTD. Это полезно, когда приложение желает отправить XML-сообщение к другому приложению, основанному на взаимно согласованной структуре DTD, или существующему в качестве серверной части того же приложения для конструирования документов XML. При использовании этих классов приложения Java могут проверять правильность, создавать и печатать XML-документы, которые основаны на DTD. Генератор классов работает вместе с Oracle XML Parser, который анализирует DTD и передает анализируемый документ генератору классов (рис. 7.9).

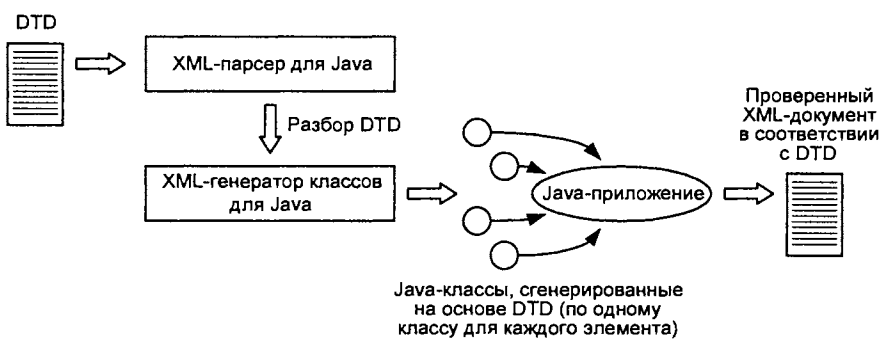


Рис. 7.9. Схема XML-генератора классов

Пример использования XML-генератора классов

Представленный ниже пример показывает, как генератор классов XML для Java может использоваться в целях обработки DTD и генерации классов для DTD-элементов. Кроме этого, демонстрируются программные методы применения классов элемента для создания полноценного XML-документа.

Входной DTD

Приводимый файл описывает служащих и имеет имя employee.dtd. Он является входным для генераторов классов. Здесь DTD определяет, что корень XML-документа — это EMP, и элемент строки — EMP_ROW. EMP состоит из одного или нескольких элементов EMP_ROWS. Каждый EMP_ROW содержит требуемый номер служащего (EMPNO), а также несколько необязательных атрибутов типа ENAME — имя служащего, JOB — тип работы, MGR — менеджер и т. д. Необязательные атрибуты сопровождаются знаком ? в описании элемента:

```
<!-- DTD for Employee Data -->
<!ELEMENT EMP (EMP_ROW)*>
<!ELEMENT EMP_ROW (EMPNO, ENAME?, JOB?, MGR?, HIREDATE?, SAL?,
                   COMM?, DEPTNO?)>
<!ATTLIST EMP_ROW ROWNO CDATA #REQUIRED>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT MGR (#PCDATA)>
<!ELEMENT HIREDATE (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
<!ELEMENT COMM (#PCDATA)>
<!ELEMENT DEPTNO (#PCDATA)>
```

Выполнение DTD для генерации Java-классов

Следующий код просто выполняет DTD и генерирует соответствующие классы для его элементов. При обработке генератором классов файла DTD создаются Java-классы для каждого элемента (EMP, EMP_ROW, EMPNO, ENAME и т. д.). Java-приложение может использовать методы, определенные в этих классах для создания действительного (допустимого — valid) XML-документа, содержащего сведения о служащих.

```
import java.io.*;
import java.net.*;
import oracle.xml.parser.*;
import oracle.xml.classgen.*;
import org.w3c.dom.Element;
```

```
public class SampleMain
{
  public SampleMain()
  {
  }
  public static void main (String args[])
  {
    // проверка аргументов
    if (args.length < 1)
    {
      System.out.println("Usage: java SampleMain "+
                         "[-root <rootName>] <fileName>");
      System.out.println("fileName\t  Input file, XML document or " +
                         "external DTD file");
      System.out.println("-root <rootName>  Name of the root Element " +
                         "(required if the input file is an external
                           DTD)");
      return;
    }
    try // открыть вложенный файл DTD
    {
      // инициировать парсер
      XMLParser parser = new XMLParser();

      if (args.length == 3)
        parser.parseDTD(fileToURL(args[2]), args[1]);
      else
        parser.parse(fileToURL(args[0]));

      XMLDocument doc = parser.getDocument();
      DTD dtd = (DTD)doc.getDoctype();
      String doctype_name = null;

      if (args.length == 3)
      {
        doctype_name = args[1];
      }
      else
      {
        // получить корневой элемент документа XML
        doctype_name = doc.getDocumentElement().getTagName();
      }

      // сгенерировать файлы Java...
      ClassGenerator generator = new ClassGenerator();

      // установить генерацию комментариев на true
      generator.setGenerateComments(true);
```

```
      // установить выводной каталог
      generator.setOutputDirectory(".");
      // установить режим проверки на true
      generator.setValidationMode(true);

      // генерировать исходный код java
      generator.generate(dtd, doctype_name);

    }
    catch (Exception e)
    {
      System.out.println ("XML Class Generator: Error " + e.toString());
      e.printStackTrace();
    }
  }

  static public URL fileToURL(String sfile)
  {
    File file = new File(sfile);
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if (fSep != null && fSep.length() == 1)
      path = path.replace(fSep.charAt(0), '/');
    if (path.length() > 0 && path.charAt(0) != '/')
      path = '/' + path;
    try
    {
      return new URL("file", null, path);
    }
    catch (java.net.MalformedURLException e)
    {
      // Can only happen if the file
      // protocol were not recognized
      throw new Error("unexpected MalformedURLException");
    }
  }
}
```

Создание действительного XML-документа из Java-классов

Следующий Java-код показывает, как сгенерированные методы могли бы использоваться. Здесь созданы два элемента строки: `emp_row1` и `emp_row2`. Элементы для каждого столбца также созданы (`empno1`, `ename1` и т. д.). Чтобы сформировать XML-дерево документа, различные элементы данных группируются путем представления всех строчных элементов в виде древесного узла. Каждый строчный элемент затем добавляется в виде узла к элементу корня документа `EMPLIST`.

```
import oracle.xml.classgen.*;
import oracle.xml.parser.*;

public class CreateEmployees
{
  public static void main (String args[])
  {
    try
    {
      EMP EMPLIST = new EMP();
      DTD dtd = EMPLIST.getDTDNode();      // get static from base
document

      // Новый служащий стр. row1
      EMP_ROW emp_row1 = new EMP_ROW(1);  // create row and set ROWNO
      EMPNO empno1 = new EMPNO("7654");
      ENAME ename1 = new ENAME("MARTIN");
      JOB job1 = new JOB("SALESMAN");
      MGR mgr1 = new MGR("7698");
      HIREDATE hiredate1 = new HIREDATE("1981-09-28 00:00:00.0");
      SAL sal1= new SAL("1250");
      COMM comm1= new COMM("1400");
      DEPTNO deptno1 = new DEPTNO("30");

      // Новый служащий стр. row2
      EMP_ROW emp_row2 = new EMP_ROW(2);  // create row and set ROWNO
      EMPNO empno2 = new EMPNO("7788");
      ENAME ename2 = new ENAME("SCOTT ");
      JOB job2 = new JOB("ANALYST ");
      MGR mgr1 = new MGR("7566");
      HIREDATE hiredate2 = new HIREDATE("1987-04-19 00:00:00.0");
      SAL sal2= new SAL("3000");
      COMM comm2= new COMM("");
      DEPTNO deptno2 = new DEPTNO("20");

      emp_row1.addnode(empno1);     // Добавить узлы дерева к стр. row1
      emp_row1.addnode(ename1);
      // ...

      emp_row2.addnode(empno2);     // Добавить узлы дерева к стр. row2
      emp_row2.addnode(ename2);
      // ...

      EMPLIST.addNode(emp1);        // Добавить стр. row1 как узел дерева
                                    // в корень EMPLIST doc
      EMPLIST.addNode(emp2);        // Добавить стр. row2 как узел дерева
                                    // в корень EMPLIST doc
```

```
      EMPLIST.validateContent();
      EMPLIST.print(System.out);
    }
    catch (Exception e)
    {
      System.out.println(e.toString());
      e.printStackTrace();
    }
  }
}
```

XML-документ, созданный Java-приложением

Java-приложение создает XML-документ, похожий на тот, что представлен ниже.

```
<?xml version="1.0"?>
<!DOCTYPE EMP SYSTEM "employee.dtd">
<EMP>
  <EMP_ROW ROWNO = "1">
    <EMPNO>7654</EMPNO>
    <ENAME>MARTIN</ENAME>
    <JOB>SALESMAN</JOB>
    <MGR>7698</MGR>
    <HIREDATE>1981-09-28 00:00:00.0</HIREDATE>
    <SAL>1250</SAL>
    <COMM>1400</COMM>
    <DEPTNO>30</DEPTNO>
  /EMP_ROW>
  <EMP_ROW ROWNO = "2">
    <EMPNO>7788</EMPNO>
    <ENAME>SCOTT</ENAME>
    <JOB>ANALYST</JOB>
    <MGR>7566</MGR>
    <HIREDATE>1987-04-19 00:00:00.0</HIREDATE>
    <SAL>3000</SAL>
    <COMM></COMM>
    <DEPTNO>20</DEPTNO>
  </EMP_ROW>
</EMP>
```

Утилита XML SQL для Java

Утилита XML SQL для Java образует набор Java-классов для:

- передачи запросов к базе данных и генерации XML-документов;
- записи XML-данных в таблицы БД.

Генерация XML из результатов запроса

Как показано на рис. 7.10, утилита XML SQL может выполнять SQL-запросы и возвращать результаты в виде XML-документа.

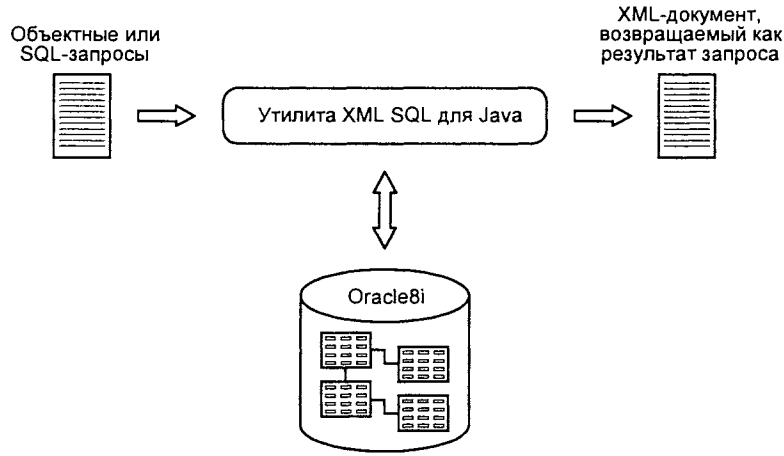


Рис. 7.10. Прохождение SQL-запроса через утилиту XML SQL

Структура результирующего XML-документа основана на внутренней структуре схемы БД и возвращаемых результатах запроса. Столбцы отображаются в элементы верхнего уровня, скалярные переменные — в элементы с текстовым содержанием, объектные типы — в элементы с атрибутами, появляющимися как субэлементы. Наборы представляются списками элементов.

Утилита может генерировать или обычное текстовое представление XML-документа, или строить в оперативной памяти XML DOM-дерево элементов. Если вы возвращаете XML-документ как ответ на запрос, то текстовое представление — самое лучшее. Вы должны использовать DOM-представление, если собираетесь обрабатывать полученный XML, например, применять XSLT-процессор или DOM-методы, чтобы осуществлять поиск в XML или изменять его.

Вы можете также воспользоваться утилитой XML SQL, чтобы генерировать DTD-описание, основанное на схеме таблицы, для которой осуществляют запрос. Разрешено применять сгенерированный DTD в качестве входного для генератора классов, который формирует набор классов, основанных на элементах DTD. Вы можете затем написать Java-код, который будет использовать эти классы, чтобы сгенерировать инфраструктуру Web-приложения. Базирующаяся на этой инфраструктуре Web-форма будет перехватывать данные пользователя и создавать XML-документ, совместимый со схемой базы данных. Эти данные могут быть потом записаны непосредственно в соответствующую таблицу базы данных.

Пример генерации XML из результирующего набора

Представленный ниже код является примером использования утилиты XML SQL для создания XML-документа из запроса. Представим, что вы посылаете следующий SQL-запрос к утилите:

```
SELECT EMPNO, ENAME FROM EMP WHERE EMPNO = 7654;
```

которая генерирует XML-документ:

```
<?xml version="1.0"?>
<ROWSET>
  <ROW id="1">
    <EMPNO>7654</EMPNO>
    <ENAME>MARTIN</ENAME>
  </ROW>
</ROWSET>
```

Пример генерации XML из результатов запроса и структурирования данных

Используя утилиту XML SQL, вы можете также проводить некоторые операции (constrains) над данными, которые представлены в XML-документе. К примеру, вы можете специфицировать максимальное число строк для возврата, количество пропускаемых строк, тип XSL-таблицы и т. д. Следующий Java-код делает запрос к СУБД и конструирует XML-файл, который содержит результаты. Запрос к СУБД прост:

```
select EMPNO, ENAME from EMP;
```

Java-код:

```
import java.sql.*;
import java.math.*;
import oracle.xml.sql.query.*;
import oracle.jdbc.*;
import oracle.jdbc.driver.*;

public class xmlquerydb
{
  public static void main(String args[]) throws SQLException
  {
    String tabName = "emp";
    String user = "scott/tiger";

    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // инициализировать соединение JDBC
    Connection conn =
      DriverManager.getConnection("jdbc:oracle:oci8:"+user+"@");
```

```
    // инициализировать OracleXMLQuery
    OracleXMLQuery qry = new OracleXMLQuery(conn,"select EMPNO,
    ENAME from "+tabName);

    // структурировать созданный документ XML
    qry.setMaxRows(2);              // установить максимальное количество
                                       возвращаемых строк
    qry.setRowsetTag("ROOTDOC");    // установить тег корневого документа
    qry.setRowTag("DBROW");         // установить тег разделителя строк
    qry.setStyleSheet("emp.xsl");   // установить таблицу стилей

    // Получить XML-документ в формате string
    String xmlString = qry.getXMLString();

    // Распечатать XML-документ
    System.out.println(" OUTPUT IS:\n"+xmlString);
  }
}
```

В результате XML-файл возвращает первую из двух строк, найденных в таблице EMP:

```
<?xml version="1.0"?>
<ROOTDOC>
  <DBROW id="1">
    <EMPNO>7876</EMPNO>
    <ENAME>ADAMS</ENAME>
  </DBROW>
  <DBROW id="2">
    <EMPNO>7499</EMPNO>
    <ENAME>ALLEN</ENAME>
  </DBROW>
</ROOTDOC>
```

Запись XML в таблицу базы данных

Как показано на рис. 7.11, вы можете использовать утилиту XML SQL для записи XML-данных в таблицу СУБД Oracle8*i*.

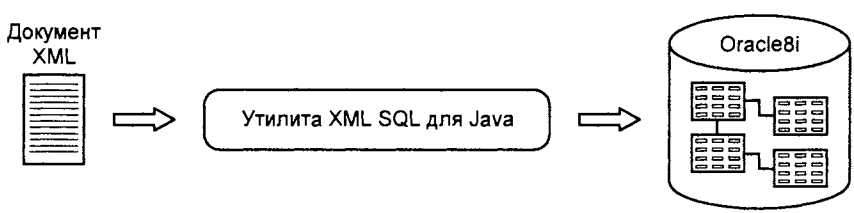


Рис. 7.11. Запись XML-данных в таблицу СУБД

В Oracle8*i* при записи XML-документов сохраняется их структура. Имена тегов элементов присваиваются именам столбцов в таблице. Списки элементов отображаются в наборы (collections). Неструктурированные данные, в виде комментариев и описаний, не могут быть отображены в таблице и сохраняются в виде CLOB в базе данных.

Пример записи XML в таблицу СУБД

Следующий код вставляет данные из XML-файла emp.xml в таблицу EMP. В этом примере принято, что XML-документ соответствует структуре таблицы EMP.

```
import oracle.xml.sql.dml.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;
import java.net.*;

public class xmlwritedb
{
  public static void main(String args[]) throws SQLException
  {
    String tabName = "EMP";          // Таблица для вставки данных XML
    String fileName = "emp.xml";     // Имя документа XML
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // Инициализировать соединение JDBC
    Connection conn =
      DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    // Вставить данные XML из файла
    // в таблицу базы данных
    OracleXMLSave save = new OracleXMLSave(conn, tabName);
    URL url = save.createURL(fileName);
    int rowCount = save.insertXML(url);

    System.out.println(" successfully inserted "+rowCount+
                       " rows into "+ tabName);
    conn.close();
  }
}
```

XSQL Servlet

XSQL Servlet — это программное средство, которое способно выполнять SQL-запросы и выводить результирующие наборы в виде XML. XSQL Servlet реализован как Java-сервлет и использует в качестве исходных данных XML-

файл, содержащий встроенные SQL-запросы. Сервлет использует парсер XML для Java и утилиту XML SQL для выполнения множества разнообразных операций.

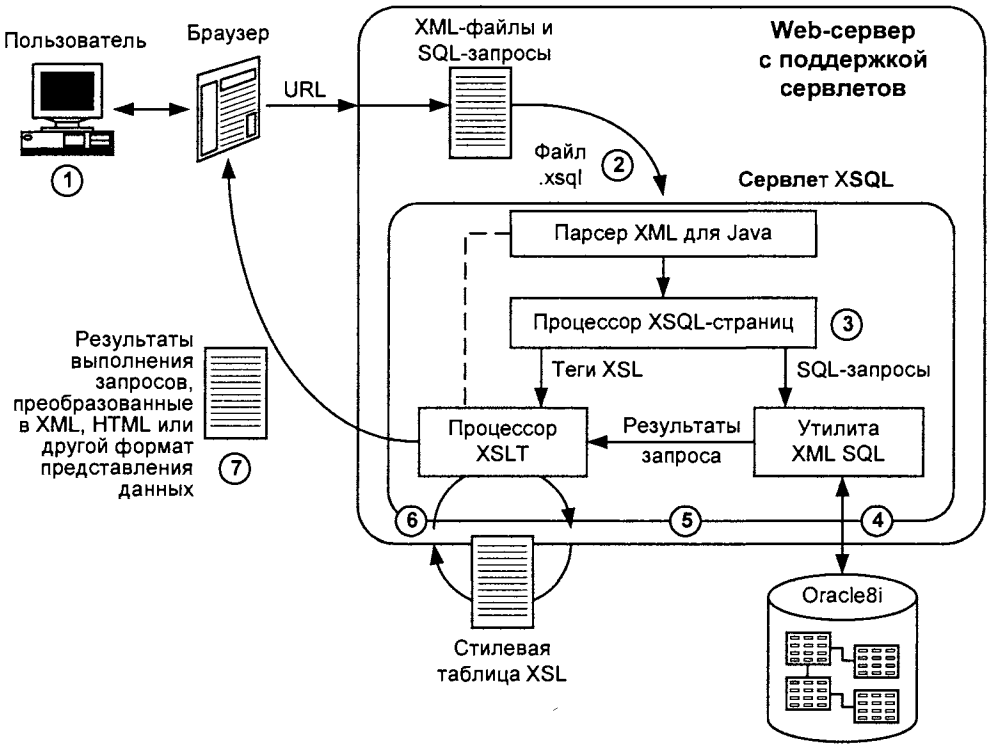


Рис. 7.12. Путь прохождения XML-данных с использованием сервлета

Вы можете выполнить этот сервлет с любого Web-сервера, который поддерживает Java-сервлеты. Рис. 7.12 демонстрирует, как данные проходят от клиента к сервлету и возвращаются обратно к клиенту. Последовательность событий следующая:

1. Пользователь вводит URL в браузере. Этот URL интерпретируется и передается к XSQL-сервлету через Java Web Server. URL содержит имя целевого XSQL-файла (xsql) и дополнительные параметры, такие как имена переменных и стилевых таблиц XSL. Альтернативно, пользователь может вызвать XSQL-сервлет из командной строки, обходя при этом браузер и Web-сервер на основе Java.
2. Сервлет передает XSQL-файл к XML-парсеру, который анализирует XML и создает API для доступа к XML-содержанию.

3. Специальный компонент сервлета (Page Processor) использует API для передачи XML-параметров и SQL-операторов (находящихся между тегами `<query></query>`) к утилите XML SQL. Page Processor также передает XSL-операторы к XSLT-процессору.
4. Утилита XML SQL посылает SQL-запрос в базу данных Oracle8*i*, которая возвращает результаты запроса в утилиту.
5. Утилита XML SQL возвращает результаты запроса в XSLT-процессор, как XML-форматированный текст. Результаты встраиваются в XML-файл в те же самые места, где находились первоначальные теги `<query></query>`.
6. По желанию, результаты запроса и любые другие XML-данные трансформируются при помощи XSLT-процессора в соответствии с заданными стилевыми таблицами XSL в приемлемую выводную форму. Данные могут быть преобразованы в HTML или в любой другой формат в соответствии с указанным стилем.
7. XSLT-процессор передает сформированный документ обратно в браузер клиента, чтобы он увидел его в том виде, как было задумано разработчиком.

Пример использования XSQL Servlet

Следующий простой XSQL-файл запрашивает таблицу служащих `EMP`. По умолчанию запрос возвращает список всех служащих. Дополнительно, вы можете сузить поиск путем добавления параметров `find = URL`, когда вызываете XSQL-сервлет из браузера. Так, указывая букву "T" как `find`-значение, вы получите только те строки, у которых фамилии в `ENAME` содержат букву "T". Вы также можете отсортировать строки, специфицируя параметр `sort = URL`. К примеру, если вы укажете в качестве параметра `EMPNO`, то сортировка будет производиться по номеру служащего.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="rowcol.xsl"?>
  <query connection="demo"
    find="%"
    sort="ENAME"
    null-indicator="yes" >
    SELECT *  FROM EMP
        WHERE ENAME LIKE '%{@find}%'
        ORDER BY {@sort}
  </query>
The XSQL file also specifies that the returned results should be proc-
essed using the XSL stylesheet rowcol.xsl. This stylesheet is as follows.
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
```

```
    <body class="page">
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="ROWSET">
  <center>
    <table border="0" cellpadding="4">
    <xsl:choose>
      <xsl:when test="ROW">
        <!-- present headings: row[1] columns -->
      <xsl:for-each select="ROW[1]">
        <tr>
          <xsl:for-each select="*">
          <th align="left">
            <xsl:attribute name="class">
            </xsl:attribute>
            <xsl:value-of select="name(.)"/>
          </th>
          </xsl:for-each>
        </tr>
      </xsl:for-each>
      <xsl:apply-templates/>
      </xsl:when>
      <xsl:otherwise>
        <tr><td>No Matches</td></tr>
      </xsl:otherwise>
    </xsl:choose>
</table>
</center>
</xsl:template>

<!-- present rows and columns -->
<xsl:template match="ROW">
  <tr>
    <xsl:attribute name="class">
    </xsl:attribute>
    <xsl:for-each select="*">
      <td>
        <xsl:attribute name="class">
        </xsl:attribute>
        <xsl:apply-templates select='.'/>
      </td>
    </xsl:for-each>
  </tr>
```

```
  </xsl:template>
</xsl:stylesheet>
```

Пример URL:

```
http://localhost/xsql/demo/emp.xsql?find=T&sort=EMPNO
```

Использование XML в Lotus Domino Server

Lotus Domino Server 5.x как нельзя лучше подходит в качестве платформы для XML.

Большинство прикладных программ чаще всего функционируют на втором уровне трехуровневой архитектуры. Клиенты и браузеры, как правило, составляют первый уровень, а третий уровень включает источники данных и собственно систему. В обязанности сервера приложений входит функционирование в качестве брокера между источниками данных на сервере СУБД ("back-end"-системы) и клиентами с внешним интерфейсом и браузерами ("front-end"-системы). Задачи XML заключаются в обмене и преобразовании данных и их интеграции из множества источников (рис. 7.13).

Рис. 7.13. Трехзвенная XML-архитектура в Domino

С точки зрения выбора прикладных платформ, перспективы Domino Application Server выглядят весьма впечатляюще по следующим причинам.

- **Производительность и масштабируемость.** Платформа, для которой вы строите приложения с использованием XML, должна быть высокопроиз-

водительной, поскольку на низкопроизводительных системах некоторые вещи могут оказаться вообще бесполезными. Масштабируемость приложений, основанных на Domino, очень высока, потому что Domino реализован для множества популярных платформ, которые способны поддерживать многопроцессорные кластерные архитектуры на Windows NT, AS/400, Sun Solaris, OS/2, HP-UX и даже OS/390.

- **Безопасность.** Сервер приложений обязан обеспечивать регистрацию пользователей с помощью различных сервисов службы управления распределенными каталогами. Кроме того, прикладные программы должны быть защищены от несанкционированного использования, даже если пользователь зарегистрирован в системе. Это необходимо для разграничения доступа к особо ценной коммерческой информации, когда она может быть получена по Интернету. Domino обеспечивает все виды шифрования информации, способен выдавать цифровые сертификаты, поддерживает механизм электронной подписи.
- **Управление памятью и запоминающими устройствами.** Сервер приложений для XML должен обеспечить механизм хранения данных или в собственном архиве, или в комбинации с внешними системами управления базами данных. Хранение данных в формате XML должно, насколько это возможно, соответствовать локальному формату и быстро преобразовываться к другим текстовым форматам в момент передачи данных. XML-базы должны также обеспечить доступ к XML-дереву через стандартные интерфейсы. Domino Server обеспечивает гибкий механизм доступа к реляционным базам данных и ERP-системам. Domino может хранить XML-документы и анализировать их различными способами (включая возможность применения собственного синтаксического анализатора XML, написанного на Java или C). Легкость использования XML в Domino основана на том, что Domino хранит сами данные и форматы их представления отдельно друг от друга, поэтому его собственная информация всегда может быть легко выражена в XML.
- **Программируемость.** Для того чтобы работать с XML-структурами, необходимы стандартные API. XML DOM является стандартным API для XML. Чтобы сделать приложение эффективным и производительным, сервер должен поддерживать наиболее популярные языки программирования для обращения к DOM. К этим средствам относятся Java, C++, C и языки управления сценариями, подобные JavaScript или VBScript. Сервер должен также поддерживать объектные расширения к объектной модели так, чтобы разработчики могли строить свои приложения с учетом определенных потребностей. Domino-сервис включает богатый набор готовых объектов и стандартных интерфейсов, изощренное управление доступом к информации, основанное на ролевом взаимодействии, машину обработки сценариев (Script Engine), виртуальную машину Java, а также объекты более высокого уровня типа форм (Forms), видов (Views) и стра-

ниц (Pages). Кроме того, Domino Designer представляет собой первоклассную интегрированную среду разработки приложений со встроенным отладчиком.

- **Поисковые средства.** Одним из наиболее важных признаков хорошего Web-сайта является наличие на нем поисковых инструментов, обеспечивающих возможности контекстного и полнотекстового поиска информации. XML может обеспечить идентификацию данных. Но вы сами должны позаботиться о навигации по этим данным. В считанные мгновения вы должны находить то, что вам нужно. Domino имеет встроенные средства поиска (Search Engine), которые можно использовать с XML.
- **Передача сообщений (Messaging).** Система передачи сообщений используется не только для электронной почты; но и для прикладных программ. Возьмите, например, почтовое сообщение, которое посылается из одной системы в другую в ответ на некоторое событие. Прикладные программы, которые используют XML, обычно применяют стандартные протоколы, типа HTTP или IIOP, для перемещения данных по сети и, конечно, SMTP-стандарт для отправки электронной почты. Инфраструктура системы передачи сообщений Domino использует SMTP как локальный почтовый протокол. Возможно, наиболее значительным обстоятельством является то, что инфраструктура системы передачи сообщений является частью Domino Server. То есть, даже если вы не работаете с системой передачи сообщений для почты, вы можете использовать ее преимущества в ваших собственных прикладных программах, например, для посылки уведомлений, копирования информации из одной базы данных Domino в другую и, вдобавок, создавать прикладные программы, которые смогут получать почту от внешних партнеров вне всякой связи с вашей локальной сетью.

Сервер приложений Domino — лучшая XML-платформа

Из всего вышеперечисленного можно сделать вывод, что Domino Server — превосходная XML-платформа (рис. 7.14). Domino обеспечивает набор основных услуг, которые позволяют прикладным XML-программам в полной мере проявлять свое волшебство: инкапсулировать данные, представлять их в виде списка объектов, перемещать по всему разнообразию систем, архитектур и типов клиентов и свободно манипулировать ими.

К достоинствам Domino-платформы можно отнести ее *открытость* и *безопасность*; она оптимизирована для поддержки быстродействующих распределенных е-бизнес приложений для крупных корпоративных систем с множеством динамичных бизнес-процессов.

Domino-сервисы включают в себя электронную почту, многочисленные протоколы связи, объектно-ориентированное хранилище данных, сертифика-

цию пользователей, глобальный каталог, встроенный поиск, поддержку кластеризации.

Набор *прикладных услуг Domino* базируется на ролевой системе управления доступом, машине поддержки сценариев, виртуальной машине Java, программных объектах высокого уровня, типа управления формами, видами, страницами во всем богатстве их комбинаций и наборов.

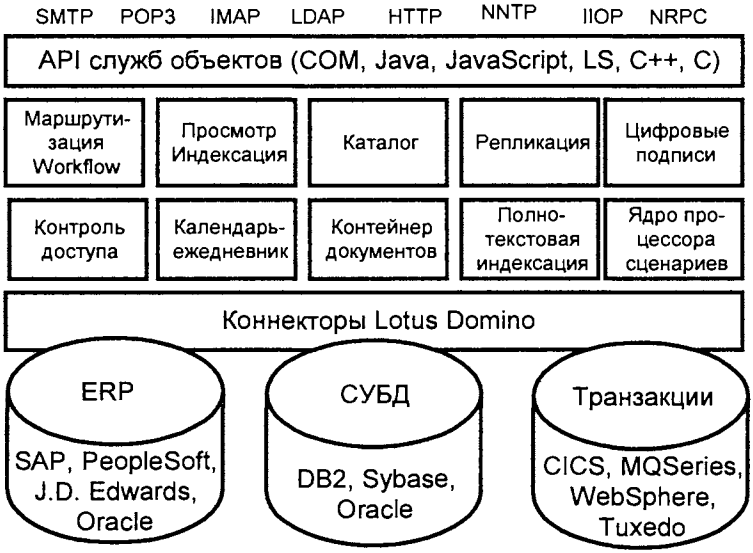


Рис. 7.14. Архитектура приложений Domino

Domino Server прекрасно приспособлен, чтобы использовать свои преимущества и эксплуатировать возможности XML и связанные с ним технологии. В конце концов, можно просто писать обычные сервлеты и агенты, которые будут выполняться на Domino-сервере и работать с XML-структурами. Кроме того, Domino-структура изначально была разработана таким образом, чтобы можно было отделить "данные" от "представлений данных", что, собственно говоря, XML и делает. Разработчики могут легко извлекать информацию, сохраненную в базе данных Domino в XML-формате. Более того, возможности Domino XML расширяются посредством LotusXSL-процессора.

LotusXSL-процессор

LotusXSL-процессор — это набор Java-классов, которые выполняются при преобразовании XML-информации в соответствии со спецификацией eXtensible Stylesheet Language Transform (XSLT). Соответствующие таблицы стилей определяют правила для преобразования исходного XML-файла в некоторую другую форму XML. В браузере разработчик мог бы использовать

XSLT и XML, чтобы сформатировать данные для клиента в соответствии с его пожеланиями. В Domino-сервере LotusXSL-процессор выполняет задачу представления информации. С помощью LotusXSL можно преобразовывать XML в обычный HTML для показа в браузерах, которые не поддерживают XML.

К примеру, если приложение имеет XML в формате:

```
<book author="Tom Jones" ISBN="12345-0392">
  <price>20</price>
</book>
```

но требуется формат:

```
<book>
  <author>Tom Jones</author>
  <ISBN>12345-0392</ISBN>
  <price>20</price>
</book>
```

вы можете вызвать LotusXSL из вашей Java-программы (агента или сервлета) и использовать подходящий метод для формирования стиля в соответствии с XSLT. Так вы можете преобразовывать данные из одного формата XML в другой. LotusXSL работает совместно с Java XML-парсером (таким, например, как IBM XML4J-парсер). Оба этих программных средства совершенно свободно можно скачать с сайта **http://alphaworks.ibm.com** и использовать в своих целях без всяких ограничений.

Комбинация XSL и XML позволяет разработчикам строить гибкие и адаптируемые к меняющимся условиям приложения в сфере электронной коммерции. Ниже приводится конкретный пример использования LotusXSL-процессора.

```
<?xml version="1.0"?>
<!— Stylesheet to do a transformation from companyA XML document type to
    companyB XML document type. —>
<xsl:transform xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
indent-result="yes">
<!— Start processing at the Document Element —>
<xsl:template match="book">
<!— Make a book element. —>
<book>
<!— Make the 'author' element. —>
<author><xsl:value-of select="@author"/></author>
<!— Make the 'ISBN' element. —>
<ISBN><xsl:value-of select="@ISBN"/></ISBN>
<!— Simply copy all other children of 'book'to the result —>
```

```
<price><xsl:value-of select="price"/></price>
</book>
</xsl:template>
</xsl:transform>
```

Как используют XML в Domino

Будущие реализации Domino, как обещают менеджеры IBM/Lotus, обеспечат наиболее полную поддержку XML. Но разработчикам нет никакой необходимости ждать, пока появятся эти реализации. Они уже сейчас могут использовать XML в прикладных программах на базе Domino. Имеется несколько способов применения XML в базах данных Lotus Domino:

- использование Domino-видов (views) и форм (forms) для экспорта XML;
- написание LotusScript или Java-агентов, которые работают с XML;
- использование Java-сервлетов для работы с XML или преобразования XML к другому формату при помощи LotusXSL-процессора.

Далее приводятся конкретные примеры совместного использования Domino и XML.

Использование видов для создания XML

В базе данных Domino, виды (views) являются представлениями основных данных, определяемых в качестве документов (рис. 7.15). Виды обеспечивают гибкий, определяемый пользователем формат, который может включать такие полезные свойства, как объединение частей документов в сворачивающиеся/разворачивающиеся категории и сортируемые столбцы с критериями отбора документов, задаваемыми пользователями при помощи формул. Используя виды, разработчики могут предопределять документы, которые будут применяться для создания XML. Поскольку каждый документ является контейнером данных, он напоминает структурированную запись базы данных и целые документы можно отбирать в результирующий набор XML, подобно тому, как это делается по оператору `SELECT` в СУБД. Этот метод хорош тем, что он использует все преимущества Domino-индексации для быстрого доступа к данным. Вы можете указывать простейшие формулы, чтобы определить, какие документы должны появиться в виде. К примеру, в базе данных для заказов книг формула

```
Status="To be ordered"
```

будет отбирать все документы, у которых поле с именем `Status` будет содержать текстовое значение `"To be ordered"`. Для создания XML, вы просто используете формулы для столбца вида, определяющие имена соответствующих данных и сами данные, которые затем должны быть извлечены из

базы данных. К примеру, первый столбец вида `Orders` (Заказы) будет формироваться по формуле:

```
"<Order><OrderItem>"+ orderItemField+"</OrderItem>"
```

А следующий столбец будет создаваться согласно формуле:

```
"<Quantity>"+quantityField+"</Quantity></Order>"
```

Когда вы добавляете столбцы, то создается XML-код. В нашем примере фрагмент XML-кода для документов по книгам, которые были уже заказаны, выглядел бы следующим образом для данного вида:

```
<Order><OrderItem>Widget</OrderItem><Quantity>2</Quantity></Order>
```

Эта строка выглядит точно так же, как если бы она была сгенерирована агентом, но никакого программирования при этом не требуется. Существует одна особенность Domino, которая делает возможным создание XML из вида при помощи свойства вида "Treat View contents as HTML." При установке указанного свойства в Lotus Designer для конкретного вида Domino не будет автоматически генерировать HTML-код, а вместо этого просто передавать текст в том виде, в каком он был создан.

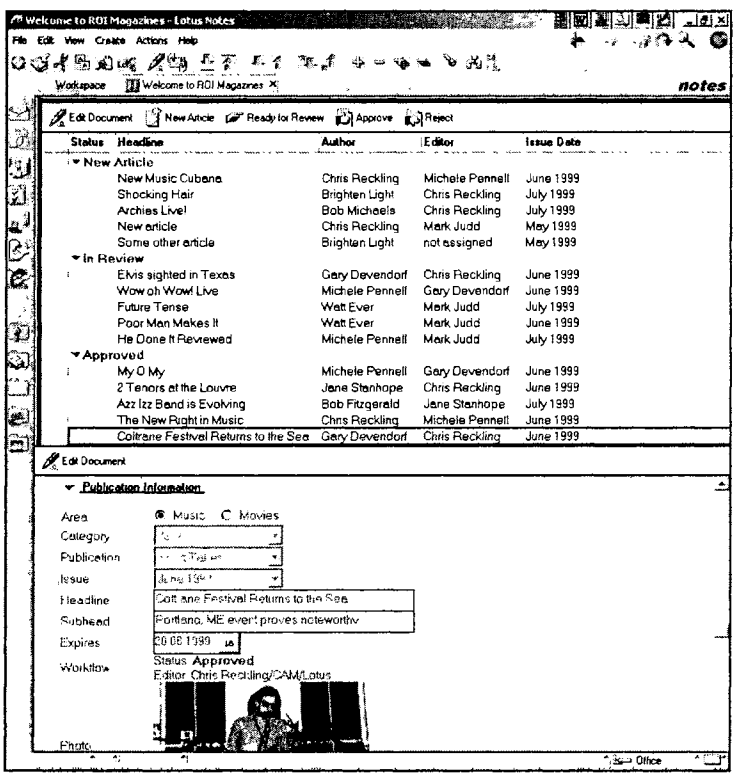


Рис. 7.15. Документ, отобранный в вид и открытый для просмотра

Создание XML-кода из простого вида при помощи формул Lotus

В следующем примере мы рассмотрим гипотетическую компанию, которая обеспечивает своим служащим заказ канцелярских товаров в режиме on-line непосредственно с их рабочих мест, оборудованных компьютерами.

1. Служащие могут заказывать товары на сумму до $50 без вышестоящего утверждения. Заказы на сумму более чем $50 должны быть одобрены непосредственным начальником служащего.
2. Служащий оформляет заказ через браузер по сети интранет, заказ обрабатывается и сохраняется в базе данных Domino (Orders On-Line database).
3. Виды (views) отслеживают состояние заказа (*ожидающий утверждения*, *скомплектованный* и *полностью обработанный*). Они разработаны так, чтобы создавать XML-файл из документов, присутствующих в данном виде. XML требуется для взаимодействия с программой обработки заказов в офисной компании, поставляющей заказанные товары.
4. Чтобы разместить заказ в офисной компании, необходимо регулярно выполнять сервлет, который запускается по графику и просматривает базу данных, обращаясь к ней через URL. Сервлет выявляет скомплектованные заказы и посылает информацию о них в формате XML в офисную компанию, тем самым размещая заказ. Когда заказ обработан, сервлет устанавливает флажок в статусе документа — *"полностью обработанный"*.
5. После этого сервер Domino посылает служащему уведомление о выполнении заказа по электронной почте, где сообщается номера заказа и ожидаемая дата поставки (рис. 7.16).

Рис. 7.16. Использование видов Domino для продуцирования XML

В этом примере, Domino обеспечивает инфраструктуру и рабочий поток (workflow) документооборота, в то время как XML является общим и стандартным структурированным языком между двумя несоизмеримыми системами, позволяя использовать технологии Domino для работы с документами и быть понятным системе, которая использует полностью отличный формат для собственных программ обработки заказов.

Использование Domino-форм для создания XML

Поскольку формы, применяемые в Domino, — это просто шаблоны для форматирования данных, содержащихся в документе, использование этих форм для создания XML-документов чрезвычайно просто. Разработчик расставляет XML-теги по полям на форме, которые соответствуют определенным полям в базе данных.

В качестве примера использования XML рассмотрим каталог для продажи книг в режиме on-line:

```
<?xml version="1.0" encoding="UTF-8"?>
<BOOK>
<bookTitle>Chess for the Master</bookTitle>
<bookCategory>Games</bookCategory>
<bookAuthor>Alice B. Charles</bookAuthor>
<bookPrice>10</bookPrice>
<bookListPrice>12</bookListPrice>
<bookISBN>0-980-38475-81</bookISBN>
<bookDatePublished>April 1997</bookDatePublished>
<bookAbstract>The authority on all the latest chess moves,
including the entire Big Blue arsenal.</bookAbstract>
</BOOK>
```

Для создания документов в XML-формате необходимо:

1. Создать новую форму или страницу.
2. Ввести декларацию типа:

   ```
   <?xml version="1.0" ?>
   ```

   Вы можете добавить желаемую таблицу перекодировки, например:

   ```
   <?xml version="1.0" encoding="UTF-8" ?>
   ```

3. Ввести XML-элементы (обычно, главный элемент и встроенные элементы).
4. Задать поля документа, для которого вы строите XML.

   Таким образом, каждый тег XML связывается с полем (рис. 7.17).
5. Выбрать в меню **Design** команду **Form Properties**.
6. На вкладке **Advanced tab** выбрать опцию **Treat document contents as HTML**.

Это свойство позволяет передавать содержимое документа Domino к HTTP без обязательного его преобразования в формат HTML.

7. Закрыть и сохранить форму.
8. Для просмотра документов, образованных на основе этой формы, создайте вид (view), который отбирает нужные вам документы по формуле.

Теперь, всякий раз, когда форма используется, чтобы сформировать, согласно ей и присутствующим в ней полям, новый документ, данные из соответствующих полей документа автоматически преобразуются в XML-код, так, чтобы XML-документ мог обрабатываться любыми серверами, синтаксическими анализаторами или XML-клиентами (рис. 7.18).

Форму можно использовать тогда, когда вам необходимо создать оригинальный XML-документ, точно соответствующий Notes-документу (в противоположность видам, которые формируют XML-структуру для всех документов сразу).

Формы также позволяют быстро и просто добавлять стилевые описания, если они необходимы, при помощи форматов CSS или XSL-таблиц. Для этого вы задаете тег, определяющий стиль, например:

```
<?xml:stylesheet type="text/xsl" href="orders.xsl"?>
```

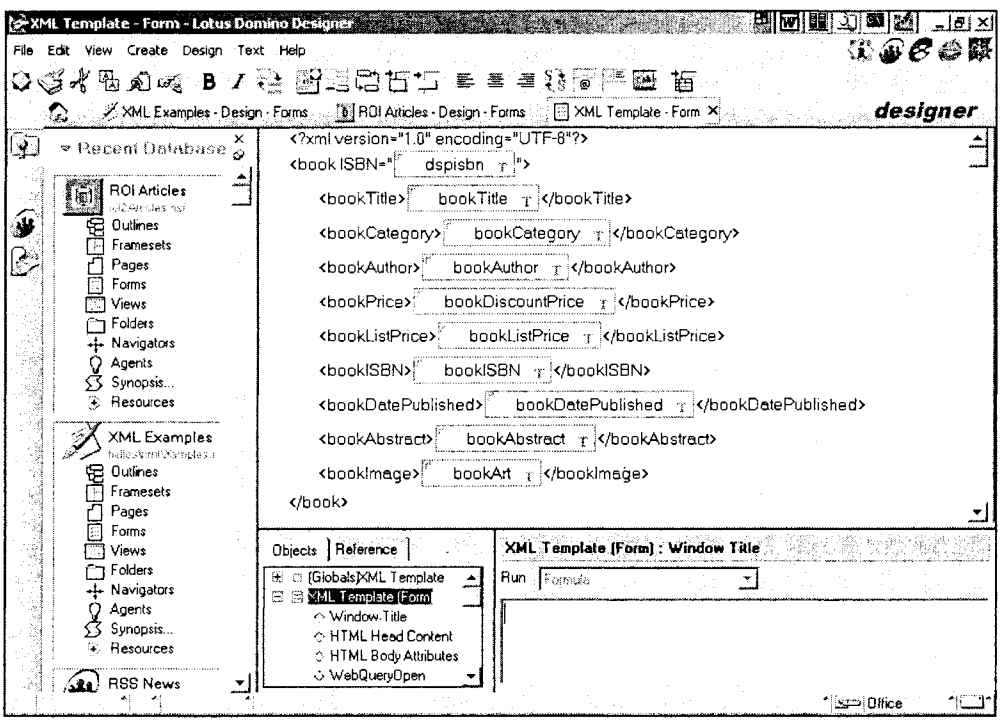


Рис. 7.17. XML-теги связываются с полями в Domino-форме

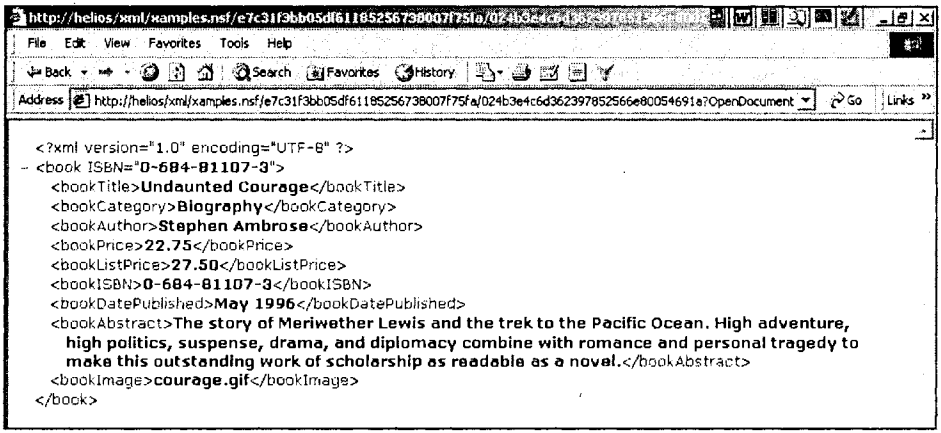


Рис. 7.18. Вывод XML-формы, показанный в Web-браузере

Создание XML при помощи агентов LotusScript или Java

Агенты хороши тем, что они обладают большой гибкостью в способах их использования. Их можно запускать по графику, они могут реагировать на некоторые события, их можно принудительно вызвать на выполнение из браузера по строке URL. Например, строка URL вида **http://server /dbname.nsf? OpenAgent** приводит к выполнению агента.

Domino-агенты весьма гибки относительно XML-кода, который они создают. Например, некое приложение может потребовать, чтобы программист написал агента, который бы генерировал XML-код из специфического набора документов и создавал его в некотором формате, типа книжных заказов в формате XML, обсужденном ранее, или, возможно, использовал формат, основанный на стандартных данных, или даже формат, который он сам, программист, придумал. Тот же самый набор документов можно использовать и в другом приложении или агенте, чтобы создать полностью отличный от первого XML-формат, необходимый для совершенно иной цели. Более того, агенты обеспечивают гибкость и относительно того, где они хранят XML-данные, которые создают, то ли в файловой системе, то ли в другой базе данных. Они могут генерировать XML-данные динамически и, соответственно, гораздо быстрее, чем, если бы они создавали XML-код, который должен быть затем сохранен.

LotusScript- и Java-агенты могут преобразовывать Domino-документы и HTML- в XML-файлы, или, наоборот — преобразовывать XML-код в Domino- и HTML-документы. В следующих примерах можно видеть оба типа преобразования: первый, когда агент используется для экспорта в XML из базы данных Domino в базу данных Web, и второй — когда XML-данные по сети перемещаются в Domino.

```
Dim s As New NotesSession
Dim db As NotesDatabase
Dim doc As NotesDocument
Dim view As NotesView
Set db = s.currentDatabase
Set view = db.GetView( "Orders")
Set doc = view.GetFirstDocument
Print "Content-type: text/xml"
Print "<?xml version="1.0" encoding="UTF-8"?>"
Print "<Orders>"
While Not ( doc Is Nothing)
  Print "<Order number="+doc.orderNumber(0)+">"
  Print "<OrderItem>"+doc.orderItem(0)+"</OrderItem>"
  Print "<Quantity>"+doc.quantity(0)+"</Quantity>"
  Print "<ItemPrice>"+doc.itemPrice(0)+"</ItemPrice>"
  Print "<shipTo>"+doc.customerName(0)+"</shipTo>"
  Print "<shippingAddress>"+doc.address(0)+"</shippingAddress>"
  Print "</Order>"
  Set doc = view.GetNextDocument( doc)
Wend
Print "</Orders>"
```

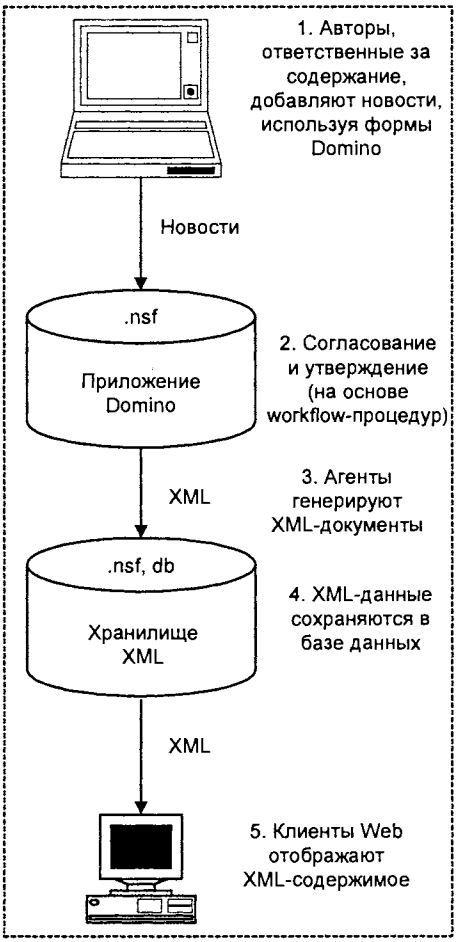
LotusScript-агент генерирует XML-код из базы данных Domino и посылает его в браузер.

Экспорт данных из базы Domino в XML-формате

В качестве очередного примера мы рассмотрим компанию, которая хочет показывать раздел "Ежедневные новости" в соответствующей рубрике своего Web-сайта. Каковы будут требования к приложению?

- Оно должно удовлетворять экспертов, которые формируют новости и новостные заголовки в соответствии со своим профилем и вводят эту информацию в систему либо в режиме off-line, либо посредством браузера, либо из Notes-клиента. Их действия при этом могут и должны быть отредактированы и одобрены другими людьми до момента окончательной публикации новостей, причем процесс согласования должен быть эффективным и быстрым.
- Абоненты новостей должны получать доступ к новостям из обычного браузера по сети Интернета.
- Новости должны сохраняться и архивироваться.

Давайте рассмотрим, как мы можем спроектировать такое приложение в виде базы данных Domino, со встроенным LotusScript-агентом, для преобразования Domino-документов в XML-код, который будет использоваться абонентами новостей. Абоненты могут применять стандартные инструмен-



тальные средства типа XML-синтаксических анализаторов, браузеров или других XML-клиентов и даже собственных прикладных программ,. способных воспринимать и трактовать формат XML.

1. Эксперты соответствующих предметных областей вносят новости в базу данных через формы Domino, и информация сохраняется в виде документов в базе данных Domino.
2. База данных Domino содержит средства для редактирования документов и их последующего утверждения (workflow-процесс).

Рис. 7.19. Использование Domino-агента для генерации XML в целях создания "Архива новостей" на Web-сайте

3. Планируемый LotusScript-агент выполняется в базе данных ежечасно, чтобы формировать свежие новости. Из новых документов в базе данных этот агент создает XML-код в общепринятом для промышленности новостном формате (News Industry Text Format). Мы будем использовать XML по следующим причинам:
   - через XML будет отображаться только та информация, в которой нуждаются абоненты новостей (мы не должны включать внутренний цикл утверждения новостных документов и связанные с этим процессом поля);
   - поскольку XML — это текстовый формат, который содержит информацию относительно своей собственной структуры ("это — заголовок, это — окончание, это — название, это — имя автора" и т. д.), мы можем создавать XML-код в стандартном формате, который дает воз-

можность другим серверам и клиентам использовать для чтения документов их собственные прикладные программы;

4. Агент использует объектные классы LotusScript, просматривает все документы и производит по ним итерацию. Агент перемещает информацию из полей документов, сохраняет их в своих собственных переменных и добавляет к ним теги XML. Затем он генерирует результирующий файл в формате XML.
5. Новостная база данных с сервера разработки реплицируется на сервер, находящийся снаружи firewall, и после этого становится доступной абонентам. Заметьте, что в этом пункте база данных уже не обязательно должна быть базой данных Domino. Это может быть любой формат базы данных на Web (рис. 7.19).

Сетевой импорт XML-данных в базу данных Domino

Разработчики могут создавать свои агенты на языках LotusScript или Java. Агенты будут перемещать XML-код по сети, анализировать его структуру и помещать информацию в базу данных для постоянного хранения и дальнейшего использования. Возьмите, например, клиента Notes, который подписался на "Ежедневные новости" из предыдущего примера. Новости, которые находятся в формате XML, могут быть взяты с Web-сервера при помощи Domino-сервлета или агента. При этом агенты свободно могут использовать сетевые сервисы Java или DCOM, которые поддерживает Domino, чтобы вызвать один из многих XML-доступных синтаксических анализаторов и разместить XML-элементы в соответствующих полях базы данных Domino.

Использование Java-сервлетов для генерации XML

Так же как и агенты, Java-сервлеты, выполняемые в Domino Server, могут генерировать XML-содержание. Разработчики приложений могут управлять XML-тегами тем же способом, как они это делают, когда используют LotusScript- или Java-агенты. К примеру, вполне допустимо запускать Java-сервлет вместо агента для создания раздела "Ежедневные новости" из предыдущего примера. Вдобавок, сервлеты могут делать и другие интересные вещи, например, различные преобразования с помощью LotusXSL-процессора перед тем, как отправить данные в браузер или на сервер. В следующем примере демонстрируется, как можно применять Java-сервлет, чтобы преобразовать информацию, полученную из различных источников, в формат XML.

Пример извлечения данных из различных источников

В этом примере вы можете увидеть, как Java-сервлет работает совместно с LotusXSL-процессором, чтобы данные, перемещенные из различного набора источников, могли обслуживаться в Domino-приложениях для различных

серверов и устройств. Каким требованиям должна удовлетворять такая программа?

- Во-первых, способность к работе через ODBC и другие средства соединения с базами данных, типа DataBase Connectors для реляционных баз данных, Notes-баз, ERP-систем и т. д., даже других форматов XML.
- Во-вторых, коммерческие приложения для мобильных клиентов должны использовать ряд различных устройств для соединения с Domino, когда такие клиенты находятся в пути, включая Notes-клиентов, браузеры и PDA[1]. Некоторые из PDA используют новый беспроводный язык разметки (WML, Wireless Markup Language). Непосредственно перед началом обслуживания мобильного клиента коммерческое приложение подключается к Domino и обновляет информацию при помощи быстрой репликации данных.

Посмотрим, как такое приложение работает.

1. Сервлет, выполняющийся в Domino, перемещает всю информацию, необходимую клиенту, из различных источников (ODBC, ERP-системы, и т. д.) и преобразует данные в XML.
2. В зависимости от того, как коммерческий служащий связан с Domino и через какое конкретно устройство, сервлет преобразует формат XML к соответствующему клиентскому формату, используя LotusXSL-процессор (рис. 7.20).

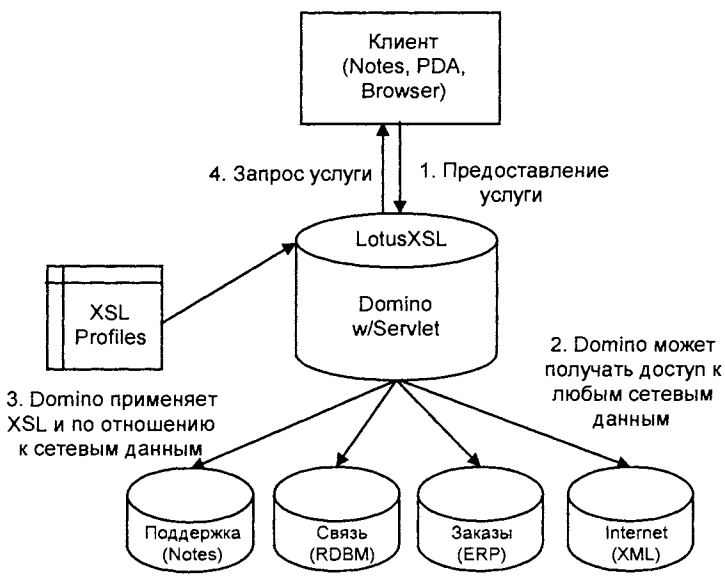


Рис. 7.20. Использование Java-сервлетов с Domino

[1] Карманный компьютер. — *Ред.*

Как Lotus использует XML в своих собственных продуктах

После краткого изложения аспектов использования XML в прикладных программах для Domino полезно взглянуть на то, как Lotus применяет XML в своих собственных продуктах.

В клиенте Notes, Domino Server V5, Lotus SmartSuite, eSuite и других продуктах Lotus использует XML-технологии для решения конкретных проблем.

Апплеты Domino для обеспечения пользовательского интерфейса

В R5 Domino-разработчики могут обеспечивать альтернативные функциональные возможности по отношению к браузерам типа MS IE и Netscape Navigator. В качестве альтернативы встроенным функциям браузеров в Domino имеются предварительно подготовленные апплеты на Java. Эти апплеты эмулируют в браузерах многие функциональные особенности Notes-клиентов, чего не могут обеспечить сами браузеры. Раньше разработчики Domino R5 должны были использовать только HTML, чтобы воспроизвести некоторые функции Lotus-клиентов в браузерах. Теперь, Java-апплеты Domino делают это более эффективно. Три апплета (views, outline и action bar) используют XML, чтобы обеспечить связь элементов дизайна и собственно данных, что означает расширение функциональных возможностей программы, поскольку HTML, все-таки, ограничен в средствах. Например, апплет, реализующий виды (views), сначала запрашивает дизайн-шаблон вида так, чтобы знать, как расположить столбцы и цвета набора, а затем читает данные из столбцов и строк. Апплеты, в силу своей специфики, выполняются на машинах клиентов, разгружая сервер и давая возможность пользователям лучше эксплуатировать их собственные ресурсы (рис. 7.21).

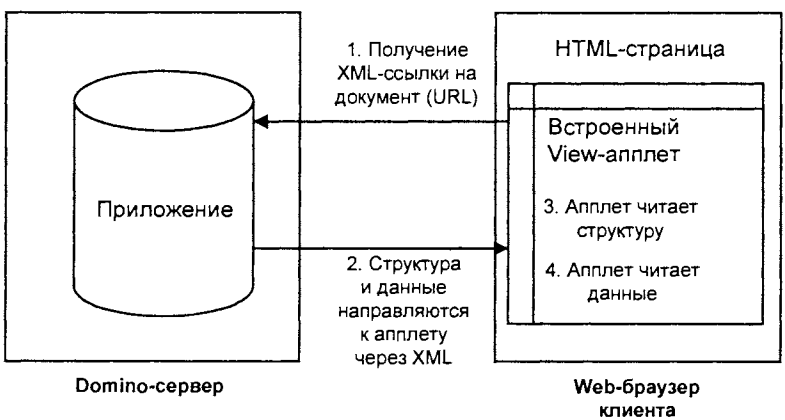


Рис. 7.21. Апплет View, связывающийся с Domino Server через HTTP

Файловые форматы в SmartSuite и eSuite

В гетерогенной сетевой среде, где не все пользователи выполняют те же самые клиентские или серверные прикладные программы, стандартный файловый формат может предоставлять возможность пользователям свободно обмениваться информацией, независимо от того, где и как файлы созданы — в собственном ли приложении, по инициативе ли удаленного пользователя, или при помощи браузера. Lotus спроектирован таким образом, чтобы обеспечить полную совместимость SmartSuite и eSuite при помощи промышленного стандарта XML. Пользователи могут открывать любой документ в SmartSuite или eSuite, и им не надо волноваться относительно происхождения этого документа, потому что он может быть прочитан и отредактирован везде и без всяких проблем. Поскольку SmartSuite и файлы eSuite используют форматы XML, разработчики могут программно обращаться к внутренней структуре документов через стандартный DOM API.

Резюме

В Lotus изначально заложен огромный потенциал XML, который уже сейчас может быть реализован, поскольку разработчики имеют необходимые средства для воплощения этой простой и мощной технологии. В самом ближайшем будущем Lotus, несомненно, будет развивать и совершенствовать эти средства и перестраивать всю свою внутреннюю инфраструктуру в соответствии с рекомендациями стандартов и спецификаций XML, принятых международными организациями. Для разработчиков это означает, что многое будет осуществляться проще и легче, а совместимость с другими прикладными системами станет обеспечиваться как по мановению волшебной палочки, поскольку в общепринятых и поддерживаемых всеми разработчиками стандартах заключена великая сила.

Microsoft и XML

Первые сообщения о том, что Microsoft заинтересовалась XML появились еще в 1996 г., а в 1997 г. язык XML стал уже частью корпоративной стратегии фирмы. Microsoft совместно с другими производителями программного обеспечения, в частности IBM/Lotus, является активным участником консорциума W3 по разработке стандартов XML. Интерес Microsoft к XML вполне понятен, потому что эта, сугубо практичная, фирма почти никогда ничего не изобретает, а только лишь приспосабливает чужие идеи к своим далеко идущим планам и всегда доводит эти идеи "до ума", в смысле извлечения из них максимальной коммерческой выгоды. Так и здесь. Вполне вероятно, что Microsoft пытается увеличить спрос на Internet Explorer 5.0 и замедлить темпы распространения Java при помощи XML. Если Microsoft добьется успеха, разработчики получат инструмент, позволяющий создавать интерфейсы для работы с базами данных на основе браузеров. Сейчас мно-

гие решают эту задачу с помощью Java. Еще одним преимуществом новой технологии станет возможность разработки форматов данных, не зависящих от поставщиков. В принципе, это самое важное, потому что формально-нейтральные данные могут существовать вечно.

На своем корпоративном сайте Microsoft открыла целый портал для XML, который одним своим видом (рис. 7.22) должен внушать ужас независимым XML-разработчикам, поскольку они видят, как много уже наработано фирмой Microsoft по XML!

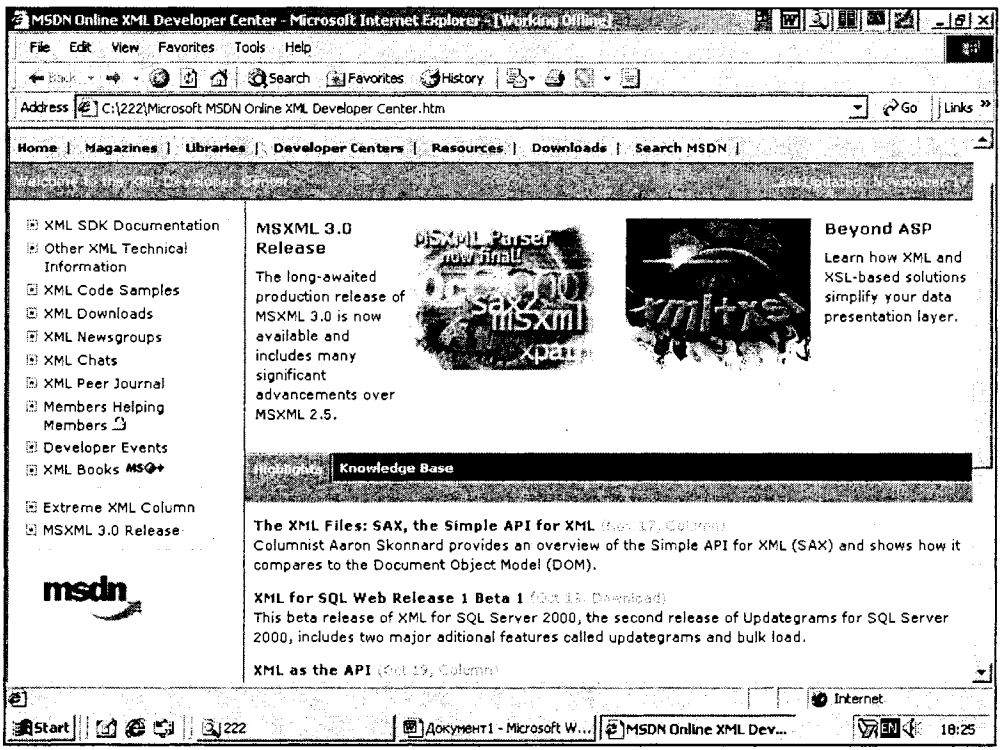


Рис. 7.22. XML-портал фирмы Microsoft

Если раньше в планах Microsoft по отношению к XML активно фигурировал лишь сервер электронной коммерции BizTalk Server, структура документов которого исключительно базируется на XML, то к концу 2000 г. корпорация публично выступила с пропагандой новой грандиозной .NET-концепции (как и всегда являющейся калькой с чужих идей; любому понятно, что .NET-концепция — это "творчески" переработанная, а вернее сказать, переваренная в желудке Microsoft архитектурная модель Java Enterprise Edition). Даже сам девиз новой .NET-концепции: "Расширить границы возможностей человека посредством ПО, работающего в любое время в любом месте на

любом устройстве" — напоминает ранние лозунги Java, когда эта, родившаяся в "пене морской", действительно звездная технология вышла на мировую сцену в своем еще младенческом возрасте. Но кто теперь помнит об этом? Все что связано с Microsoft, поражает множество людей своим грандиозным масштабом, своей огромностью, но эта грандиозность кажущаяся, по сути — это плоская беспредельность, огромное общее место. В новой концепции существует внушительная гамма серверов и обмен данными между ними может производиться при помощи универсального и нейтрального к любой архитектуре XML — зачем вам теперь Java? В табл. 7.1 приводится список этих серверов, производимых Microsoft.

Таблица 7.1. *Серверы с поддержкой XML*

Название	Функции	Назначение
SQL Sever 2000	Хранилище данных	СУБД с поддержкой XML и OLAP
Exchange 2000	Поддержка коллективной работы	GroupWare-сервер, система обмена сообщениями и система управления знаниями с поддержкой XML
BizTalk Sever 2000	Сервер бизнес-процессов	Координация бизнес-процессов на базе XML-схем в рамках одной или нескольких организаций в гетерогенной среде
Commerce Server 2000	Электронный бизнес	Сервер поддержки электронной коммерции
Application Center 2000	Сервер масштабирования приложений	Поддержка серверных кластеров и распределение заданий по множеству процессоров
Internet Security & Acceleration Server 2000	Брандмауэр и прокси-сервер	Защита и повышение производительности корпоративной сети
Mobile Information Server 2001	Сервер для мобильных клиентов	Доступ к корпоративным ресурсам с различных мобильных устройств
Tahoe	Корпоративный портал	Публикация и поиск информации в корпоративных интрасетях

Подробное рассмотрение всех этих технологий Microsoft не входит в задачу автора настоящей книги. Скажем только, что, как и всегда, у Microsoft (отдадим ей в этом должное) прекрасно проработаны все пользовательские интерфейсы и обеспечен полный комфорт при работе с фирменными продуктами. Например, пользовательские интерфейсы в BizTalk с использованием XML организованы превосходно.

BizTalk Editor

Этот инструмент предназначен для создания и редактирования схем спецификаций документов. Схема документа — это XML-документ, в котором описывается формат документа, его структура и типы данных. На рис. 7.23 показано окно **BizTalk Editor**, а на рис. 7.24 — XML-схема документа.

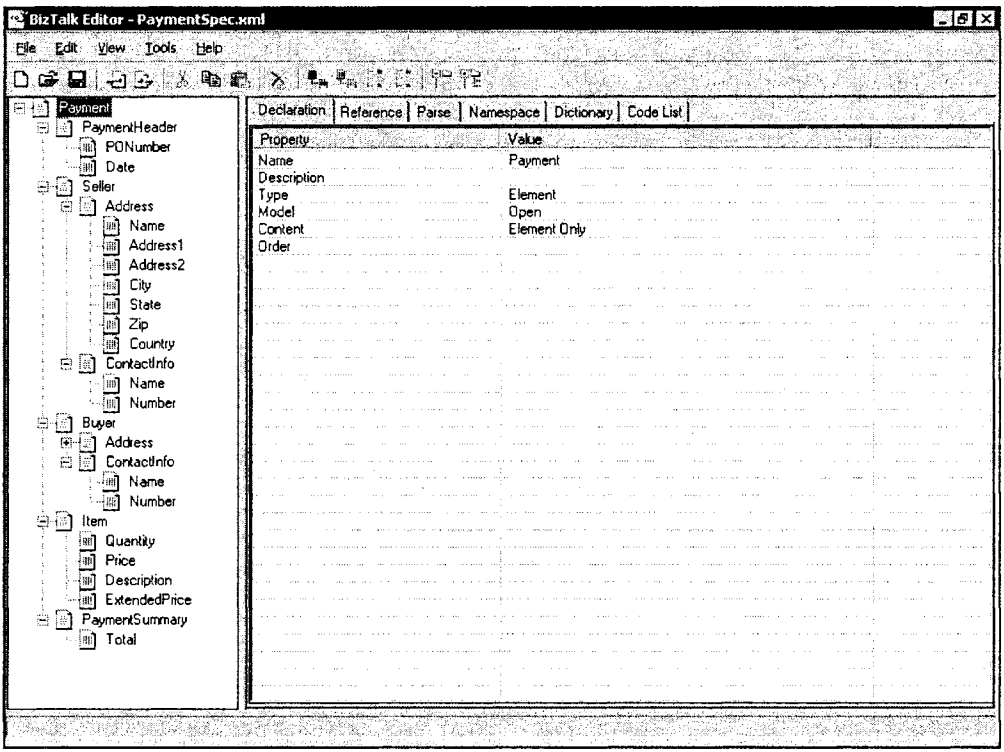


Рис. 7.23. Окно BizTalk Editor

Ранее уже говорилось о назначении парсеров. У Microsoft, разумеется, имеется свой парсер — Microsoft XML Parser, — работающий по модели DOM.

В BizTalk Server для преобразования документов существует специальный инструмент BizTalk Mapper, который использует стандартный механизм преобразования — XML Transform (XMLT). В нем на языке XML описывается преобразование документа из одного формата в другой.

С помощью утилиты BizTalk Mapper можно задать способ преобразования документа из одной схемы в другую. Созданную с помощью BizTalk Mapper схему преобразования можно поместить в репозиторий BizTalk Server и затем использовать при описании соединений (рис. 7.25).

d:\Temp\CononShppingAdvic.xml - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites History

Address d:\Temp\CononShppingAdvic.xml Go Links

```
<?xml version="1.0"?>
    <Schema name=PaymentSpec" b:root_reference="PaymentSpec" b:standard="XML"
     xmlns="urn:schemas-microsoft-com:xml-data"
     xmlns:b="urn:schemas-microsoft-com:BizTalkServer">
     <b:SelectionFields />
    <ElementType name="WareHouseItemDetail" content="empty" model="closed"
     <description> Warehouse Item Detail </description>
     <b:RecordInfo />
    <AttributeType name="UnitsShipped" d:type="r4" d:maxLength="10" d:minLength="1">
    <description> Numeric Value of Units Shipped </description>
      <b:FieldInfo/>
```

Done Local intranet

Рис. 7.24. Схема XML-документа

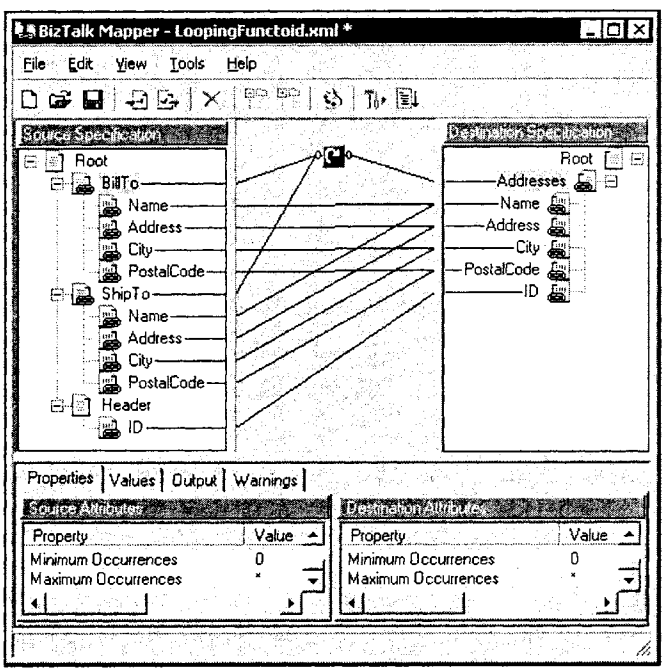


Рис. 7.25. Окно **BizTalk Mapper**

BizTalk Mapper

Планы у Microsoft в отношении .NET весьма далеко идущие. Вот как раскрывает суть .NET-плана исполнительный директор и президент Microsoft Стив Балмер в интервью газете "eWeek":

> *"Это — модель программирования следующего поколения. Специалистам она предоставляет все преимущества языка, а пользователей выводит на качественно новый уровень работы. В рамках этой инициативы будет создан интерфейс пользователя, разработаны новые серверные услуги, предложены средства клиентского контроля за разнообразными Web-узлами. Из чего она складывается? Из связующего ПО и интерфейсов пользователя, которые будут внедрены в оконечные устройства, серверы и Интернет. Вместо того чтобы работать с каждым приложением вроде Exchange и программы бухгалтерского учета по отдельности, я воспользуюсь новым сервисом, который согласует их между собой с помощью языка XML и предложит в рамках одной услуги. Задачу согласования как раз и будет решать платформа .Net. Мы хотим наладить активные каналы связи с независимыми компаниями, которые специализируются в согласовании и интеграции отдельных продуктов".*

Однако это пока всего лишь планы, в то время как технологии Java в тесной связке с XML, СУБД и многофункциональными системами типа Oracle, IBM WebSphere и Lotus Domino Server уже реально работают.

Приложение к главе 7. Интернет-ресурсы и источники

- XML:
  - **http://www.xml.org/**
  - **http://www.xml.com/**
  - **http://www.jguru.com/faq/XML**
- JSP:
  - **http://java.sun.com/products/jsp**
  - **http://www.jguru.com/faq/JSP**
- DOM-спецификация:
  - **http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001 /java-language-binding.html**
  - **http://www.w3.org/TR/REC-DOM-Level-1/**
- Java-XML:
  - JSR 31:

    ttp://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.html
  - IBM XML Master: **http://alphaworks.ibm.com/tech/xmas**
  - Purple Technology XDB: **http://www.purpletech.com/xdb/**
  - XBeans: **http://www.xbeans.org/**
- Tomcat:

  http://jakarta.apache.org/
- Java Server Web Development Kit (JSWDK):

  http://java.sun.com/products/servlets
- XML4J, IBM's DOM parser:

  http://alphaworks.ibm.com/
- For XMLEntryList:

  http://www.purpletech.com/code/src/com/purpletech/xml/XMLEntryList.java
- For Resin, by Caucho (XPath processor):

  http://www.caucho.com/
- "Understanding JSP Model 2 Architecture," Govind Seshadri (JavaWorld, December 1999):

  http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Поддержка различных языков

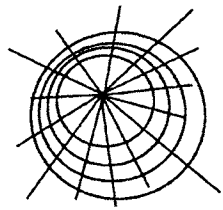
- Sun's internationalization home page:
  http://web2.java.sun.com/products//jdk/1.1/docs/guide/intl/index.html
- List of valid language codes (ISO-639):
  http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt
- List of valid country codes (ISO-3166):
  http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
- List of Java-supported encodings:
  http://java.sun.com/products/jdk/1.1/docs/guide/intl/encoding.doc.html
- Download the latest JavaServer Web Development Kit:
  http://java.sun.com/products/jsp/download.html

Книги

- XML in IE5 Programmers Reference
  http://www.amazon.com/exec/obidos/ASIN/1861001576/asptoday
- Instant DHTML Scriptlets
  http://www.amazon.com/exec/obidos/ASIN/186100138X/asptoday
- XML Applications
  http://www.amazon.com/exec/obidos/ASIN/1861001525/asptoday
- XML Design And Implementation
  http://www.amazon.com/exec/obidos/ASIN/1861002289/asptoday
- W3C XML Specification
  http://www.w3.org/TR/1998/REC-xml-19980210
- Microsoft Scriptlets Home Page
  http://msdn.microsoft.com/scripting/scriptlets/default.htm
- "Writing COM Objects with Scripting Languages"[1]
  http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/techart/msdn_xmlscripts.htm

[1] Статья. — *Ред.*

ГЛАВА 8



Что такое GNU, Open Source и Интернет-сообщество программистов

Поскольку Интернет является феноменальным явлением культуры и занимает особое место в истории человеческой цивилизации, он устанавливает свои собственные законы внутри Сети — этого самоорганизующегося информационного пространства, на которое невозможно воздействовать никакими декретами и указами, из какой бы страны и от какого бы правительства (пусть даже и мирового!) они не исходили. У Интернета нет и не может быть центральной администрации, очень трудно устанавливать железные правила: что можно делать и чего нельзя делать в Интернете. Интернет — это пространство без границ, по которому свободно перемещается информация. В связи с этим очень часто возникают вопросы о защите авторских прав на информацию, о способах ее тиражирования и распространения, о патентовании и лицензировании программ, программных технологий, программных инструментов и т. д. Социальные последствия тотальной информатизации общества сейчас предугадать не дано никому, но вот что касается программной составляющей этой информатизации, то тут бушуют программистские страсти. Дело в том, что и сам Интернет был создан некогда программистами, ими же была создана первоначальная структура и базовое программное обеспечение Интернета, базирующееся на группе основополагающих стандартов и протоколов, таких как TCP/IP, DNS, SMTP, HTTP, HTML. Это программное обеспечение могли использовать все желающие без каких бы то ни было ограничений, и оно непрерывно совершенствовалось, благодаря свободному обмену идеями. Параллельно этому процессу мир вступил в эру персональных компьютеров, которые изначально создавались именно как персональные (Personal Desktop): с персональной операционной системой, без всякой связи с внешним миром, с возможностью обмена электронной информацией только через компактные переносимые дискеты. Первые версии MS DOS не содержали в себе ни средств электрон-

ной почты, ни сетевых протоколов, ни даже возможностей обмениваться с другим компьютером элементарными файлами, если только они не были предварительно скопированы на дискету. Это уже потом появились локальные сети, сетевые адаптеры, файловые серверы. Производство программ для компьютеров превратилось в серьезный бизнес. Как только за дело взялись бизнесмены, для программистов всего мира наступила совсем иная жизнь. Производство информации и связанных с нею программных компонентов стало управляться не столько техническими соображениями, сколько законами конкуренции. Свобода пользования программным обеспечением стала ограничиваться лицензией на его приобретение. Разработчики программ начали пользоваться своими авторскими правами, как средством заставить пользователей придерживаться многочисленных правил. Естественно, с этим не захотели смириться многие.

Одним из них был Ричард Столлман (Richard Stallman, **http://www.stallman.org/**), профессор Массачусетского технологического института, известный программист, создатель популярного текстового редактора EMACS. Столлман обратился через Интернет к программистам всего мира, в котором предложил чрезвычайно простой и в то же время невероятно сложный проект, реализация которого могла вновь им вернуть утраченную свободу. Суть этого проекта, который получил название GNU (в соответствии с хакерской традицией как рекурсивный акроним выражения "GNU's Not Unix" ("GNU — это не Unix")), заключалась в том, что необходимо переписать все программы, на которые установили свои права разработчики и монополисты системного программного обеспечения. Переписать все заново! И открыть исходные тексты для всеобщего обозрения и бесконечного совершенствования. Согласно идеям Столлмана и его последователей производители машин имеют право на получение прибыли от продажи каждого экземпляра машины, потому что каждая машина воссоздается заново и на нее затрачивается строго определенное количество труда и материалов. Однажды созданная программа, на которую также затрачивается труд, потом может воссоздаваться в бесчисленном количестве копий (почти без всяких затрат), и за каждую копию невозможно требовать уплаты суммы, равной стоимости первоначального оригинала. Системное программное обеспечение должно быть бесплатным, всеобще доступным и существовать в исходных кодах. Только так можно предотвратить тиранию, которая может угрожать человечеству в недалеком будущем. Столлман опубликовал в Сети свой начальный анонс проекта (**http://www.gnu.org/gnu/initial-announcement.ru.html**) и Манифест GNU (**http://www.gnu.org/gnu/manifesto.ru.html**), где доказывал все преимущества использования свободных программ и программ с открытым исходным кодом (Open Source). Не откажу себе в удовольствии дать обширные выдержки из этих выдающихся документов.

"Первое, что вы делали, решив воспользоваться компьютером, — это обещали отказать в помощи своему соседу. Взаимопомощь категорически воспрещалась.

Правило, которым руководствовались владельцы коммерческого ПО, звучало так: "Если вы поделились программой со своим соседом — вы пират. Если вы хотите что-либо в ней изменить — обращайтесь к нам".

Вывод о том, что социальный уклад коммерческого ПО, т. е. уклад, который запрещает вам делиться или обмениваться программами, антиобщественен, неэтичен, попросту несправедлив, некоторым из читателей может показаться абсурдным. Но как еще можно назвать уклад, базирующийся на разделении общества и осознанно оставляющую пользователей без помощи? Читатели, которым данный вывод кажется абсурдным, могут принять социальный уклад коммерческого ПО как данность или оценивать ее с позиций компаний, предлагающих коммерческие программы. Сторонники данного подхода долго и упорно убеждали людей в том, что такой подход является единственно возможным. Когда поставщики коммерческого ПО говорят о "соблюдении своих прав" или "необходимости остановить пиратство", на самом деле это всего лишь прикрытие их действительных намерений. По сути, за всеми этими лозунгами скрывается ряд предположений, правомерность которых никаким сомнениям не подвергается; предполагается, что общественность примет эти предположения как само собой разумеющееся. Давайте разберемся.

Первое предположение состоит в том, что программные компании имеют несомненное естественное право собственности на свое программное обеспечение и, как следствие, власть над всеми своими пользователями. (Если бы это было естественным правом, тогда вне зависимости от того, какой вред оно бы ни несло, мы не могли бы против этого протестовать.) Интересно отметить тот факт, что Конституция США и правовые традиции опровергают эту точку зрения; авторское право не является естественным правом, а является искусственно созданной, навязанной правительством монополией, которая ограничивает естественное право пользователей на копирование программ. Еще одно предположение заключается в том, что единственная важная характеристика любой программы — то, какую работу она вам позволяет выполнять; то есть мы, пользователи компьютеров, не должны заботиться о том, в каком именно обществе оно позволяет нам жить.

Третье предположение — мы никогда не получим полезную программу, если не согласимся на то, что разработчик имеет власть над всеми пользователями ее программы. Это предположение казалось правдоподобным до тех пор, пока движение за свободно распространяемые программы не продемонстрировало, что мы можем получить достаточно полезное ПО, не заковывая при этом себя в цепи рабства.

Если мы не принимаем эти предположения за истину и рассматриваем данные проблемы, исходя из общепринятой морали, при этом на первое место ставя интересы пользователей, мы придем к совсем иным выводам. Пользователи должны иметь право свободно изменять программы в соответствии со своими нуждами и свободно обмениваться программами, поскольку взаимопомощь — основа общества."

Несмотря на раздающиеся из противоположного лагеря ядовитые смешки, прекраснодушные идеи Столлмана были подхвачены программистами, множество из которых решили последовать за своим вдохновенным проповедником всеобщего программистского освобождения. Еще в первых своих статьях по проекту Столлман предлагал использовать в качестве всеобщего прототипа операционной системы Unix и это послужило ключом к успеху — триумфальное шествие по миру свободной операционной системы Linux приобретает все более грандиозный и глобальный масштаб, сметая на своем пути все и всяческие препоны и границы. Столлман последовательно продолжал курс, изложенный в Манифесте, и организовал Фонд Свободного Программного Обеспечения (Free Software Foundation), который занимается разработкой свободных программ.

Опять вернемся к цитированию Ричарда Столлмана:

"Первые шаги

Незадолго до начала Проекта GNU я услышал о Free University Computer Kit, также известном как VUCK (голландское слово 'свобода' начинается с буквы V). Это компилятор, способный работать со многими языками, в том числе Си и Паскаль, и создавать исполняемые модули, которые можно было запускать на машинах самого разного типа. Я обратился к автору компилятора с просьбой разрешить мне использовать его для реализации своего проекта.

Он прислал мне язвительный ответ, заявив, что университет свободный, а компилятор — нет. Поэтому я решил, что первой программой, которую я создам в рамках Проекта GNU, станет многоязычный, многоплатформенный компилятор. В этот момент я осознал, что должен написать свой компилятор полностью, с самого начала. Им стал компилятор, теперь известный как GCC.

Может ли любой свободно воспользоваться программой?

Если программа, выходя из-под пера своего автора, становится свободно распространяемой, это необязательно означает, что каждый, кто имеет ее копию, может свободно ее распространять. Например, общедоступное ПО, т. е. такое, на которое не распространяется авторское право, является свободно распространяемым; но любой человек может на его основе создать коммерческую программу. Точно так же многие свободно распространяемые программы защищены авторским правом, но распространяются по условиям простых разрешающих лицензий, которые допускают создание модифицированных коммерческих версий.

Классическим примером тому является оконная система X Window System. Эта система, созданная в MIT и выпускаемая как свободно распространяемое ПО с разрешающей лицензией, вскоре стала широко использоваться различными компьютерными компаниями. Они добавляли X к своим коммерческим Unix-системам только в двоичном виде и продавали его с тем же самым соглашени-

ем о неразглашении. Эти экземпляры X оказывались свободно распространяемым ПО не более чем сами Unix-системы.

Разработчиков X Window System такая ситуация не удивила, поскольку они предполагали, что это произойдет. Целью была не свобода, а "успех", определяемый как "большое число пользователей". Они не заботились о том, насколько свободными окажутся пользователи и хотели лишь, чтобы их было много.

Это привело к парадоксальной ситуации, где различные способы определения свободы давали различные ответы на вопрос: является ли эта программа свободно распространяемой? Если судить о том, каким образом эту свободу определяют условия распространения X Window, можно сказать, что она относится к свободно распространяемому программному обеспечению. Но если попытаться выяснить, какой свободой обладает ее конкретный пользователь, можно с полным правом назвать эту систему коммерческой. Большинство пользователей X работали с коммерческими версиями, которые поставляются с Unix-системами, а не со свободно распространяемой ее версией.

Linux и GNU/Linux

В 1991 году Линус Торвальдс разработал Unix-совместимое ядро и назвал его Linux. В 1992 году объединение Linux с неполной системой GNU позволило предложить свободно распространяемую операционную систему. (Их объединение само по себе оказалось делом нелегким.) Именно благодаря Linux мы может работать с версией системы GNU.

Мы называем эту систему GNU/Linux, чтобы подчеркнуть, что она представляет собой именно сочетание системы GNU с Linux в качестве ядра.

Будущие трудности

Мы доказали свою способность создавать широкий спектр свободно распространяемых программ. Это не значит, что мы непобедимы и всемогущи. Существует ряд вопросов, из-за которых будущее свободно распространяемого ПО выглядит достаточно неопределенным; решение этих вопросов потребует стойкости и долготерпения, возможно в течение нескольких лет. Этим вопросам посвящены следующие четыре раздела.

Секреты оборудования

Производители аппаратного обеспечения стремятся сохранить в секрете спецификации оборудования. Такой подход сильно затрудняет создание свободно распространяемых драйверов так, чтобы Linux и XFree86 могли поддерживать новое оборудование. На сегодняшний день полная свободно распространяемая система существует, но завтра она может не оказаться таковой, если мы не сможем поддерживать новые модели компьютеров.

Эта проблема решается двумя способами. Программисты могут выполнять обратный инжиниринг, чтобы выяснить, как именно поддерживать аппарату-

ру. Остальные могут ограничить свои усилия оборудованием, которое поддерживается свободно распространяемыми программами; по мере роста наших рядов, секретные спецификации будут обречены на забвение.

Обратный инжиниринг — огромная работа; есть ли у нас программисты, решимости которых хватит для того, чтобы ее выполнить? Да, если мы непоколебимо верим в то, что свободно распространяемое ПО — это дело принципа, а драйверы, не относящиеся к такому типу программ, использоваться не должны. Готовы ли мы потратить дополнительные деньги и время, чтобы иметь возможность использовать свободно распространяемые драйверы? Да, если решимость получить свободу станет стремлением большинства."

Что из себя представляет GNU сегодня? Это грандиозная коллекция совершенно бесплатного ПО, которое подразделяется на четыре категории (см. **http://www.fsf.org/gnulist/production/index.html**).

- **Системное ПО** (сюда входят операционная система Linux, многочисленные компиляторы почти со всех языков программирования, графические оконные интерфейсы, системы обработки текстов и т. д.). Следует сказать, что независимо от GNU существует еще одна свободно распространяемая операционная система BSD, которая разработана в Калифорнийском университете в Беркли (UC Berkeley).
- **Средства разработки** (сюда входят помимо тех же компиляторов различные интегрирующие среды программирования и удобные средства проектирования и разработки).
- **Утилиты и библиотеки** — сотни различных утилит и библиотек.
- **Приложения** — различные приложения, например, MC-командер (типа программы FAR, но для Unix), которые поставляются в исходных кодах и при сборке компилируются для любой программно-аппаратной платформы с использованием средств GNU.

Помимо этих перечисленных программных средств к свободно распространяемому ПО можно отнести и великолепный Apache и средства программирования на Java для любых программных систем и архитектур и много других новинок, например — оффисный пакет Star Office, который может работать под Linux, а по своей функциональности приближается к MS Office.

Недавно всемирно известная компания Sun выступила с новой весьма впечатляющей инициативой. Sun решила опубликовать интерфейсы Java API для XML Messaging и Java API для XML Parsing (см. **http://java.sun.com/jdc**).

Вместе с интерфейсом Java API для XML Data Binding они станут ключевыми для платформы Java 2. Больше того, Sun рассекретила исходные коды ядра Solaris 8 и набора вспомогательных приложений, которые доступны по адресу **http://www.sun.com/solaris/source**.

Совместно с сообществом разработчиков Apache Software Foundation компании Sun и Kodak готовят свободно распространяемое средство Java для создания масштабируемой векторной графики, основанной на XML.

В мире Linux также происходят большие перемены. Появляется все большее число некоммерческих дистрибутивов. Например, Debian GNU/Linux поддерживает шесть процессорных архитектур: Intel x86, Motorola 680x0, DEC Alpha, Sun SPARC, ARM и Power PC — и базируется на версии ядра Linux 2.2.17, библиотеке glibc 2.1.3 и графической подсистеме Xfree86-3.3.6. Для архитектуры i386 после перекомпиляции ядра обеспечивается поддержка симметричной многопроцессорности.

С помощью этих средств можно построить любые сколь угодно сложные и высоконадежные системы, полностью отвечающие современному уровню. Интересно отметить, что оригинальный дистрибутив Debian GNU/Linux создавался в сети Интернет виртуальным коллективом в числе 500 программистов добровольцев и нескольких тысяч пользователей со всего мира. Координировали работы сотрудники компаний IPLabs и АНО "Институт логики".

История проекта GNU и других ему подобных проектов не завершена. Наоборот, она только еще начинается. К этому увлекательному и благородному делу подключается все большее число ярких, неординарных личностей. Например, наш соотечественник Николай Безруков в своей статье "Разработка программ с открытыми исходниками как особый вид научных исследований", опубликованной в Интернете (**http://www.linux.zp.ua/translates/open-src-as-acad-rsrch.html**) пишет:

"Прежде всего, я хотел бы подчеркнуть, что Интернет может значительно сократить стоимость обеспечения пользователей некоторыми типами программ, такими как ОС, компиляторы и утилиты. Интернет предоставляет возможность создания бесконечного количества доступных удаленному пользователю абсолютно идентичных копий компьютерных программ, мультимедиа-презентаций либо интересных дискуссий по электронной почте.

Общепризнанным является тот факт, что близкая к нулю цена копирования компьютерной программы создает существенные различия между программами и другими потребительскими продуктами. Эти различия в некоторой степени игнорируются или подавляются в общепринятой модели "коробочного" распространения программ. Я думаю, что разработка программы подобна созданию прикладной теории. Для меня предпочтительнее классифицировать программирование как особую разновидность научной деятельности или, по крайней мере, как нечто очень близкое к ней. Второе важное различие между программами и другими товарами состоит в том, что небольшие модификации легко осуществимы. Программы легче модифицировать. Это еще одна общая черта программ, сближающая их с научными теориями.

Как в науке, так и в программировании участники занимаются этой деятельностью не ради денег. Большинство разработчиков программ с открытыми ис-

ходниками делают это в погоне за мечтой, а не для пополнения своих банковских счетов. Их мотивировки могут различаться, но, попросту говоря, способности многих выдающихся программистов используются не полностью на тех должностях (часто в корпоративной среде), на которых им приходится работать. Некоторые нуждаются в специфическом инструменте и обнаруживают, что или он недоступен, или слишком дорог, и поэтому пытаются создать его сами. Опять-таки, они преследуют свою мечту, а не конкурента.

Благодаря Интернету сегодня стало возможным создавать виртуальные коллективы разработчиков программ, подобно научным коллективам, когда несколько исследователей, заинтересованных конкретным феноменом, создают виртуальное научное сообщество для решения вопроса или научной проблемы.

Структура такого коллектива и распределение обязанностей в нем могут меняться динамически. Программа будет создаваться не отдельной личностью или группой, а глубоко взаимосвязанным коллективом заинтересованных в этой проблеме участников. Я хотел бы подчеркнуть, что эти виртуальные коллективы, возможно, напоминают неформальные сообщества ученых прошлого[1]*, которые обменивались рукописными посланиями, делясь друг с другом достижениями, идеями и критикой".*

С этими словами трудно не согласиться. Мы не знаем каким станет общество в будущем и что нас ждет, но мы знаем, что программирование начинает оказывать на мир все более и более возрастающее влияние. Профессия программиста сегодня наиболее востребованная. Чтобы в совершенстве овладеть этой профессией, необходимо очень много знать и уметь. Автор этой книги выражает огромную благодарность своим читателям, которые смогли прочесть ее до конца и, главное, поприветствовать те сотни и тысячи авторов Интернета, с чьими мыслями, трудами и уникальным опытом создания программ можно не только детально ознакомиться, но и опробовать их на практике, использовать в лекционных курсах, творчески развивать, не говоря уже о благородных задачах просвещения и ради достижения тех самых свобод, о которых так ярко и вдохновенно пишут отцы-основатели Free Software Foundation и других творческих союзов в программистском сообществе, основанном на взаимном доверии и взаимопомощи.

---

[1] Работавшие над проблемой, общаясь по переписке. — *Авт.*