Лекция 4. ASP - язык серверных сценариев

4.1. Обзор клиент-серверных технологий в Интернет.

Как мы уже убедились статические HTML страницы существенно проигрывают в сравнении с DHTML, поэтому мимо разработка языка сценариев не могла пройти и компания Microsoft. Так в противовес JavaScript разработанному компанией Netscape появился VBScript, разработанный. компанией Microsoft Эти языки используются на стороне клиента, то есть генерируют объекты на основании HTML-страницы на стороне клиента в окне его браузера.

Данный подход имеет некоторые недостатки:

- сложность доступа к данным, находящимся на сервере
- высокие требования к аппаратным возможностям клиента
- высокая нагрузка на сеть, так как сервер вынужден передавать много лишней информации, не имея информации о конкретных запросахпользователя

Эти проблемы возникают в связи с тем, что использование только скриптов заставляет решать все проблемы пользовательскую машину. Противоположный подход – решение все проблем на сервере так же имеет много недостатков, основной из которых связан с перегрузкой сервера при большом количестве запросов. Типичным примером такого подходя является CGI.

Что же такое CGI (Common Gateway Interface, интерфейс общего шлюза)?

По сути CGI — способ взаимодействия Web-программ с браузером пользователя. Поэтому под CGI-программами понимают программы, написанные на любом языке программирования, способного выполняться на Web-сервере, включая C, C++, Visual Basic или даже командные языки операционных сред (например, C Shell). Но большинство CGI-программ пишется на языке Perl.

Perl (Practical Extraction and Report Language) является одним из наиболее гибких языковых средств, служащих для программирования интерфейсов CGI. Изначально Perl предназначался для обработки больших объемов данных и генерации отчетов по обработке этих данных (как явствует из его названия). За последние несколько лет Perl превратился в полнофункциональный язык программирования. Изначально созданный исключительно для работы под управлением операционных систем семейства UNIX, Perl теперь совместим с такими ОС, как Amiga, MS-DOS, OS/2 Warp, VMS, Windows NT, Window 95 и Macintosh.

Таким образом, CGI осуществляет запуск Web-приложения на стороне сервера, данные выводимые этим приложением на стандартный вывод и являются HTML-страницей, посылаемой клиенту. Всякий раз, когда клиент инициирует выполнение CGI-приложения, Web-сервер выполняет отдельную его копию (instance). Проблема заключается в том, что для каждого запроса клиента запускается копия Web-приложения на сервере, что резко сокращает производительность сервера при больших и средних нагрузках

ASP и PHP

Понятно, что решить вышеперечисленные проблемы позволит только технология, в которой бы сочетались интерактивные возможности скриптов и мощные возможности серверных приложений. При этом серверная часть не должна быть самостоятельным приложением, что бы не создавать свой instance при каждом обращений.

В последнее время все большую популярность получают два средства создания интерактивных Web-страниц — ASP и PHP, работающие именно таким образом . Основным их достоинством является возможность формирования страниц на основании интерактива «клиент-сервер». Сами же программы, написанные на ASP (Active Server Pages — активные серверные страницы) и PHP (Personal Home Page), настолько просты, что программирование с их помощью доступно даже неискушенным.

PHP часто еще называют препроцессором гипертекста (Hypertext Preprocessor). По сути PHP серверный (выполняющийся на стороне сервера) мультиплатформный язык описания сценариев, встраиваемый непосредственно в HTML-код. В настоящее время PHP интенсивно используют более полумиллиона доменов Всемирной компьютерной сети, он распространяется на правах freeware и его можно свободно скачать с сайта разработчика http://www.php.net/. Основу синтаксиса PHP составляют язык программирования C, Java и Perl. Целью создания языка является разработка динамически генерируемых страниц в кратчайшие сроки.

Несмотря на то что PHP — прекрасная альтернатива ASP, мы остановимся на последнем. Сравнивая эти два средства, решающие по сути схожие задачи, следует отметить переносимость первого (PHP) в отличие от второго (ASP) и специальную «заточку» ASP под создание гибких и удобных интерфейсов к базам данных. Это включает использование ActiveX Data Objects (ADO). Колоссальная поддержка структурированного языка запросов к базам данных SQL является мощнейшим средством, используя которое разработчик может не переучиваясь, работать напрямую с базами данных привычным образом. ASP поддерживает работу со всеми базами данных, соответствующими стандарту ODBC.

Говоря простыми словами, Active Server Pages — это обычные страницы, которые содержат скрипты, выполняющиеся на сервере наряду с обычным HTML-кодом. Принцип работы прост: после того как «серверный» код обработан сервером, результирующая страница, содержащая только клиентский код (HTML, JavaScript, VBScript), посылается клиенту. Код, выполнявшийся на стороне сервера, увидеть в окне браузера невозможно — вы видите лишь результат его работы.

ASP работает как под управлением Windows (NT,2000,2003) сервера (необходимо установить Windows Server и Web-сервер с поддержкой ASP — Microsoft's Internet Information Server [IIS]), так и под управлением других операционных систем. В последнее время компания Chili!Soft разработала версии ASP для следующих операционных систем:

- *Linux*
- *Windows*
- *Solaris*
- *AIX*
- *HP-UX*

ASP становится совместимым со все большим числом операционных систем.

```
<%@ LANGUAGE="VBSCRIPT" %>
 <HTML>
 <HEAD>
 <TITLE>ASP test</TITLE>
 </HEAD>
 <BODY>
 <%= "Good day! It is now " & time %>
 </BODY>
 </HTML>
```

Написание CGI (Common Gateway Interface)-программ требовало особой осторожности и весьма высокой квалификации от программистов и администраторов, возникали сложности при отладке большинства приложений написанных на C, C++ или Perl. Когда компания Microsoft выпустила 3-ю версию своего Web-сервера (Internet Information Server), в начале 1997 года был создан принципиально новый метод написания серверных приложений.

4.2. Общие сведения о ASP

Вы пишете программу и складываете в файл на сервере. Браузер клиента запрашивает файл. Файл сначала интерпретируется сервером, на выходе производится HTML-код. Этот HTML посылается клиенту. Файлы с программами имеют расширение .asp. Файлы asp – это обычные текстовые файлы, содержащие исходные тексты программ. Файлы делаются с помощью любого текстового редактора. Каталог, в котором размещены файлы asp должен иметь права на

выполнение, так как сервер исполняет эти файлы, когда браузер их запрашивает. Собственно программы пишутся на любом скриптовом языке, который установлен в системе. По умолчанию поддерживаются VBScript и JavaScript. Можно доустановить другие (например, Perl). Если ничего специально не указывать используется VBScript. В дальнейшем будем ссылаться только на него. Программные фрагменты заключаются в скобки <% %>. Можно ставить открывающую скобку в начале файла, закрывающую – в конце, все что между ними – программа на Visual Basic'e.

VBScript

В VBScript есть все нормальные конструкции структурного программирования (if, while, case, etc). Есть переменные (описывать не обязательно, тип явно не задается). Поддерживаются объекты. Работа с ними обычная – Object.Property, Object.Method. Есть ряд встроенных объектов (Request, Response, Session, Server, Connection, Recordset). Можно доустанавливать другие компоненты (скачивать, покупать, программировать), например для работы с электронной почтой.

Понятия "экран", куда можно выводить данные нет. Все, что надо показать пользователю, выбрасывается в выходной поток на языке HTML. Для упрощения вывода существует объект **Response**. Вывод осуществляется с помощью метода **Write**.

```
Response.Write("<h2>Hello, world!</h2>").
```

Так производится запись во внутренний буфер объекта Response. Когда скрипт заканчивает работу, весь буфер выдается клиенту. Надо заметить, что клиент получает "чистый" HTML, таким образом программы на ASP не зависят от клиентского ПО, что очень важно. Если внутри выводимой строки нужно использовать кавычку, кавычка удваивается. Метод **Response.Redirect** перенаправляет браузер на другую страницу. Чтобы им пользоваться, нельзя до него на странице использовать Response.Write.

Ввод и вывод на ASP

Программа на ASP не может явно спросить пользователя о чем-то. Она получает данные из других страниц, либо через URL. Передаваемые параметры помещаются во входной поток и доступны через объект **Request**. Чтобы передать переменную **var** в программу **test.asp**, надо написать:

```
test.asp?var=abc
```

Чтобы из программы получить значение этой переменной, надо написать:

```
var = Request("var")
```

Несколько переменных разделяется знаком &:

```
test.asp?var1=abc&var2=def
```

Кроме того, чтобы задавать параметры в URL, можно воспользоваться формами HTML. В вызывающей странице пишем так:

```
<form method="get" action="test.asp">
<input type=text name="var1" value="default">
<input type=submit value="Submit Form">
</form>
```

При этом пользователь увидит форму из одного поля ввода (var1), в нем будет значение по умолчанию "default". Кнопка "Submit Form" завершает заполнение формы и передает все переменные на test.asp (action). Если method="get", переменные передаются через URL (test.asp?var1=default&var2=var2value). Если method="post", передаются вместе с запросом так, что внешне передача переменных не заметна. В вызываемой программе безразлично, какой метод использовался (почти). Если у вас нет специальных аргументов за метод GET, используйте метод POST.

Взаимосвязь между отдельными страницами в ASP

Обычно сервер WWW не хранит состояние приложения, т.е. все запросы взаимонезависимы, и нет стандартного способа понять, что несколько запросов пришли от

одного и того же пользователя. Но это очень нужно для разработки полноценных приложений и является одной из главных проблем разработки Web-приложений.

Один из методов решения этой проблемы - **cookies**. Пользователю при первом обращении выдается специальный идентификатор, после этого браузер пользователя предъявляет этот идентификатор при каждом обращении, и сервер может распознать, что это тот же самый пользователь. Пользователь может отключить cookies, в этом случае этот метод не работает.

ASP, используя cookies, предоставляет программисту более простое средство - объект **Session** (сессия). Сессия стартует, когда новый пользователь обращается к любому asp-файлу приложения. Сессия заканчивается при отсутствии активности пользователя в течение 20 минут, либо по явной команде. Специальный объект **Session** хранит состояние сессии. Туда можно записывать переменные, которые доступны из любой страницы в этой сессии. Записать данные в этот объект можно просто:

```
Session("var") = var
```

Считать потом еще проще:

```
var = Session("var")
```

Сессия, таким образом, – это еще один метод передачи данных между страницами. Одна страница пишет данные в сессию, другая – берет потом оттуда.

Наряду с объектом Session существует объект **Application**. Если сессия создается для каждого нового пользователя, до Application существует в единственном экземпляре, и может использоваться всеми страницами приложения.

```
Application("var") = var
var = Application("var")
```

Использование внешних компонент

Если на сервере установлены дополнительные компоненты, их можно использовать из ASP. Стандартные объекты (например из библиотек ADO (Connection и Recordset) и Scripting (Dictionary, FileSystemObject)) доступны всегда. Установка новой компоненты обычно состоит в копировании dll-файла в каталог на сервере и ее регистрации с помощью программы regsvr32.exe

Создать экземпляр объекта можно так:

```
Set var  = Server.CreateObject("Class.Object")
```

Class.Object указываются в документации на компоненту. В переменной var запоминается ссылка на созданный экземпляр объекта. Когда объект не нужен, ссылку нужно обнулить с помощью команды:

```
Set var = Nothing
```

Пожалуйста всегда обнуляйте все ссылки на объекты, когда они больше не нужны. Теоретически это должно происходить автоматически при завершении процедуры/страницы, однако в стандартной сборке мусора есть определенные "проблемы".

В остальном использование компоненты зависит от самой этой компоненты.

Работа с базами данных

Из ASP можно легко и просто работать с любыми базами данных. Это делается через две промежуточные технологии: ODBC и ADO.

ODBC позволяет организовать доступ к любым базам данных через унифицированный интерфейс с помощью языка SQL. Специфика конкретных СУБД учитывается при помощи специальных драйверов БД. Такие драйверы существуют для всевозможных СУБД (в частности SQL Server, Oracle, Access, FoxPro). Поддержка ODBC обеспечивается на уровне операционной системы Windows. Настройка – через Control Panel/ODBC. Базовым понятием является источник данных или data source. Источник данных – это совокупность сведений о базе данных, включая ее драйвер, имя компьютера и файла, параметры. Чтобы пользоваться базой надо создать источник данных для нее. Важно, чтобы источник данных был "системным", в отличии от "пользовательского". После этого надо лишь знать имя источника данных. [В настоящее время ODBC отступает перед натиском технологии OLE DB. На практике это однако практически

ничего не изменяет. Вместо имени источника данных нужно использовать Connection String, в которой указывается имя ODBC-драйвера и все его параметры.]

ADO – это совокупность объектов, доступных из ASP, позволяющих обращаться к источнику данных ODBC [или OLE DB]. Фактически нужны лишь 2 объекта – **Connection**, представляющий соединение с базой данных и **Recordset**, представляющий набор записей, полученный от источника. Сначала необходимо открыть соединение, потом к нему привязать Recordset, потом, пользуясь методами Recordset'a, обрабатывать данные. Вот пример:

```
<%
Dim Conn, RS, strSQL, strOut
strOut = ""
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Data-Source-Name"
Set RS = Server.CreateObject("ADODB.Recordset")
strSQL = "SELECT * FROM AGENTS ORDER BY USER_ID"
RS.Open strSQL, Conn
RS.MoveFirst
strOut = strOut & "<P>Here is the data:"
strOut = strOut & "<TABLE BORDER=""1"">"
strOut = strOut & "<TR><TH>USER_ID</TH><TH>Name</TH></TR>"
Do While Not RS.EOF
        strOut = strOut & "<TR>"
        strOut = strOut & "<TD>" & RS.Fields("USER_ID") & "</TD>"
        strOut = strOut & "<TD>" & RS.Fields("NAME") & "</TD>"
        strOut = strOut & "</TR>"
        RS.MoveNext
Loop
strOut = strOut & "</TABLE>"
strOut = strOut & "<HR>"
strOut = strOut & "That's it!"
RS.Close
Set RS = Nothing
Conn.Close
Set Conn = Nothing
Response.Write strOut
%>
```

Если команда SQL не возвращает данных, recordset не нужен, надо пользоваться методом Conn.**Execute**(SQL_COMMAND).

Методики программирования, советы

Описание переменных

VBScript - очень нетребовательный к программисту язык. Так он не требует описывать переменные и не содержит явных типов данных. Все переменные принадлежат одному типу **Variant**. Из-за отсутствия описаний могут произойти очень трудно обнаруживаемые ошибки. Одна опечатка может стоить полдня поисков.

Однако, есть возможность явно потребовать описания переменных. Для этого **первой** строкой в ASP-файле нужно написать **Option Explicit**. После этого обращение к переменной, которая не была объявлена с помощью **Dim**, вызывает ошибку с указанием номера строки.
Кстати, где расположены описания Dim в процедуре - совершенно не важно. Они могут стоять как до использования переменной, так и после, и даже в цикле. Видимо они отрабатываются препроцессором. Явно задать тип переменной с помощью **Dim Var as Typ**, как в Visual Basic, все равно нельзя.

Чередование ASP/HTML

Если нужно выдать большой кусок HTML, можно не пользоваться Response.Write. Если в asp-файле встречается кусок текста вне скобок <% %>, он трактуется просто как HTML, который надо вывести. Пример:

```
<% If var="VAL" Then %>
      Дорогой друг!
      …
```

```
        <% Else %>
              Извините, вам сюда нельзя
              …
        <% End If %>
```

Обработка ошибок

Для отслеживания ошибок используется специальный объект **Err**. Он устанавливается в ненулевое значение, если предыдущая команда породила ошибку. Ее можно проверять с помощью If, и таким образом реагировать на ошибки. Чтобы из-за ошибки не прерывалось выполнение программы, в начале нужно включить команду

```
On Error Resume Next
```

Включение других файлов

Можно выносить повторяющийся код в отдельный файл, и подключать к разным другим по мере необходимости с помощью команды **include**. Это очень удобно, если вы хотите вынести повторяющийся код в отдельный файл и использовать снова и снова в разных страницах:

```
<!--#include file="filename.inc" -->
```

Обработка форм

Если надо что-то спросить у пользователя и на основании этого что-то сделать, в простейшем случае создается два файла: один с формой, второй – с ее обработчиком. Обработчик выполняет все действия. Пример:

```
form.htm:
          <form method="post" action="run.asp">
          <input type=text name="var1" value="default">
          <input type=hidden name="var2" value="var2value">
          <input type=submit value="Обработать">
          </form>
run.asp:
          var1 = Request("var1")
          var2 = Request("var2")
          …
          что-то делаем
          …
          Response.Write("все хорошо!<br>”)
```

Рекурсивная обработка форм

Удобный метод состоит в том, чтобы сбор данных и обработку осуществлял один и тот же файл. Для этого пишется asp, в котором есть разные разделы. Специальная переменная отвечает за выбор раздела при запуске. Пример:

```
test.asp:
<%
Action = Request("Action")
If Action = "" Then
   ' Action не заполнен – значит надо показать форму
   Action = "FORM"
End If
Select Case Action
  Case "FORM"
  ' Показываем форму, Action="PROCESS", т.е. второй раз пойдет по другой ветке
  %>
    <form method="post" action="test.asp">
    <input type=hidden name="Action" value="PROCESS">
    <input type=text name="var1" value="default">
    <input type=submit value="Обработать">
    </form>
<%
  Case "PROCESS"
        ' Обрабатываем
    var1 = Request("var1")
    var2 = Request("var2")
    …
     что-то делаем
    …
```

```
    Response.Write("все хорошо!<br>")
End Select
%>
```

Переадресация

Очень легко написать на ASP скрипт, который будет производить некоторые расчеты, и в зависимости от результатов переадресовывать браузер на разные URL (например, подставлять нужный баннер). Делается это так:

```
Response.Redirect "somewhere.html"
```

Только надо следить, чтобы до выполнения команды redirect ничего не было записано в Response (даже коментарии HTML).

Электронная почта

Одна из часто встречающихся задач – отправить электронную почту с Web-страницы. На первый взгляд, можно просто написать

```
<FORM ACTION="mailto:user@host.com">
```

Но это приводит к тому, что при отправке формы делается попытка на клиентской машине запустить почтовую программу и создать новое сообщение с данными формы. Если это не получается (почтовая программа не настроена, пользователь не отправил почту, и т.д.) - письмо и не будет отправлено. Гораздо надежнее работает серверное решение.

Для этого существуют внешние компоненты, есть и бесплатные. Например, компонента **Jmail** от Dimac. Все, что для нее нужно – это адрес SMTP-сервера. Вот пример ее использования:

```
On Error Resume Next
Set Mailer = Server.CreateObject("JMail.SMTPMail")
If err<>0 Then
  Response.Write("Невозможно создать объект")
Else
  Mailer.ServerAddress = SMTP_SERVER
  Mailer.Sender = EMAIL
  Mailer.Subject = "Service: Delivery order"
  Mailer.AddRecipient ADMIN_EMAIL
  Mailer.Body = "-----------------------------------" & chr(10)
  Mailer.AppendText "body text"
  Mailer.Execute
  If err<>0 Then
    Response.Write("Mail error")
  Else
    Response.Write("Mail send Ok")
  End If
End If
Set Mailer = Nothing
```

4.3. Интерфейс к базе данных с помощью ASP

Постановка задачи

Предположим, перед нами стоит задача создания газетного сайта, причем некоторые из его читателей должны иметь право пополнять его новыми статьями, снабжаемыми фотографиями непосредственно с самого сайта и без программирования. По сути, необходимо создать базу данных всех статей, подготовить несколько HTML-форм для загрузки статей на сервер и отображения результата.

Я почти уверен, что подобная задача актуальна применительно практически к любому сайту, тем более что подобным образом работает большинство гостевых книг (guestbook).

Создание и подготовка базы данных

Прежде всего создадим базу данных статей, для чего: запустим приложение Microsoft Access; любым из известных способов создадим новую базу данных. Назовем ее «Articles»; в созданной базе данных создадим таблицу с именем, например «Articles»; пользуясь

инструментом «Конструктор», определим поля нашей таблицы и типы принимаемых ими значений (рис.1);

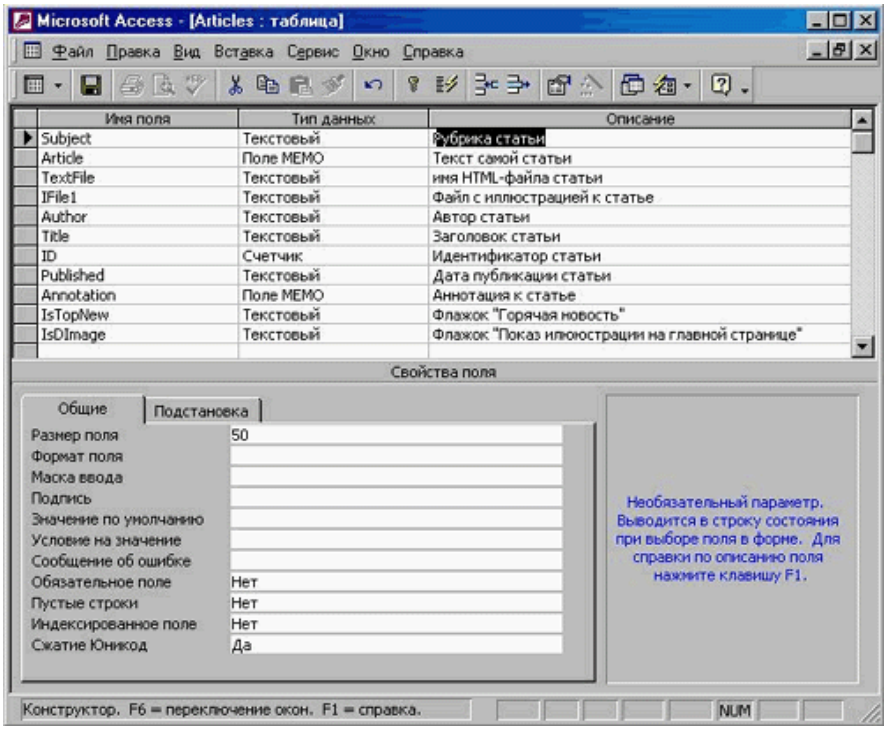
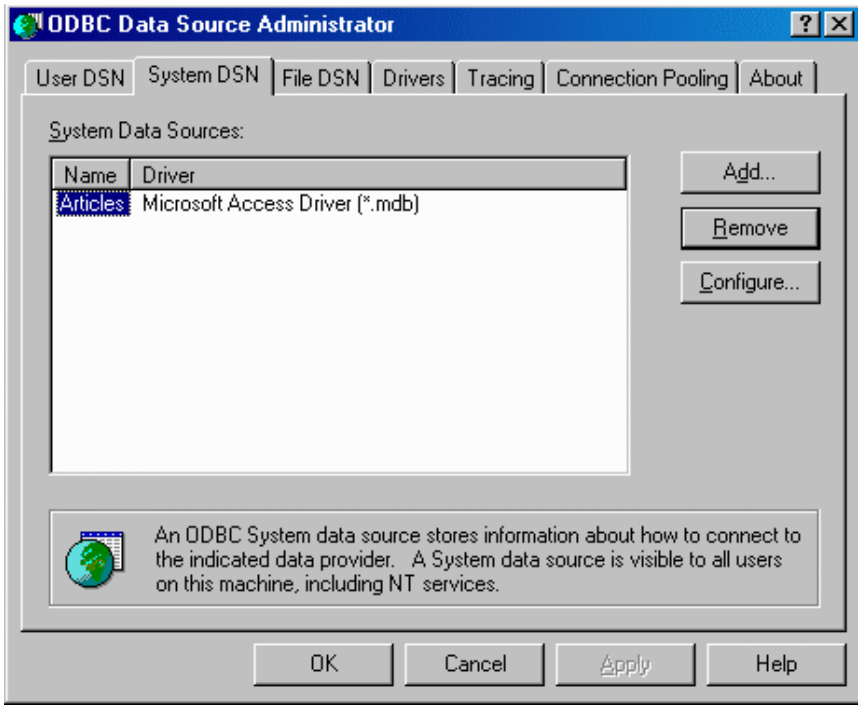


Рис.1.

заполним таблицу несколькими статьями в соответствии с созданными полями; сохраним базу данных в файле «ArticlesDB.mdb».

Далее необходимо прописать нашу базу данных в соответствующем разделе источников данных системы, для этого: запустим программу-конфигуратор источников данных (Data Sources ODBC) — Start->Settings->Control Panel->Administrative Tools->Data Sources ODBC; перейдем во вкладку «System DSN» и создадим новый источник данных, нажав на «Add…»; в появившемся списке драйверов выберем драйвер баз данных Microsoft Access — «Microsoft Access Driver (*.mdb)» и нажмем на «Finish»; в строке «Data Source Name» зададим имя нашей базы данных, например «Articles» (это то имя, по которому мы в дальнейшем будем обращаться к ней);



нажмем на «Select…», выберем подготовленный нами файл «ArticlesDB.mdb» и нажмем «OK».

Оформляем главную страницу (index.asp)

```
<html>
<head>
<title>Наш газетный сайт - первые шаги в ASP</title>
</head><body>
```

```
<center>
<p><font size="3" color="#000000"><b>Главная страница</b>
<! -- Определим основные линки -->
<center>
<a href="http://localhost/lr3/SearchForm.asp"> Поиск статьи</a><br>
<a href="http://localhost/lr3/UploadForm.asp"> Загрузка статьи</a><br>
</center>
<table BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="640" >
<tr><td WIDTH="640" BGCOLOR="#FFCC99" valign = top>
Список всех статей нашей базы данных:
</td></tr>
<! -- Наша таблица будет содержать всего одну колонку,
напишем в ней пока заголовок -->

<%
Set db = Server.CreateObject("ADODB.Connection")
db.Open "Articles"
sSQL = "SELECT * FROM Articles"
Set rs = db.Execute(sSQL)
Cnt = 0 'Заводим счетчик статей
Do While NOT Rs.EOF 'Пока в базе есть статьи (начало цикла)
%>
<tr> <!--Новый ряд в таблице -->
<td> <!--Новая ячейка -->
<%

If rs.Fields ("Article").value <> "No Text" Then
Link = "<a href= http://localhost/lr3/ArtTempl.asp?id=" _
& rs.Fields("ID").value & ">" & rs.Fields("Title").value & "</a>"
End If

Response.Write Link & "<br>"
Response.Write "<i>By " & rs.Fields("Author").value & "</i><br>"
ANN = rs.Fields("Annotation").value
If ANN <> "NA" Then
  Response.Write ANN
End If
%>

</td> <!--Конец ячейки -->
</tr> <!-- Конец ряда -->

<%
Rs.MoveNext 'Переход к следующей записи в базе статей
Cnt = Cnt + 1 'Увеличение счетчика статей
Loop 'Конец цикла
db.Close
Set db = Nothing
%>

<p><font size="3" color="#000000"><i>
<%
D = Date()
D = FormatDateTime(D,2)
Response.Write "Сегодня "& D 'Отображаем на экране в текущей позиции
Response.Write ". В базе  - " & Cnt & " статей"
%>
</i></font>
</font></p>
</table>
</center></body></html>
```

Теперь давайте разберемся. Особый интерес вызывает переменная rs. Для искушенных программистов сразу скажу — это указатель. Однако в ASP с целью облегчения работы

начинающих указатели маскируются. Здесь не встретишь громоздких С'шных конструкций, типа указатель на указатель… Однако сделано это так искусно, что гибкость программирования при этом не теряется, нет лишь прямой работы с указателями, а только работа с помощью специальных функций, скрывающих от программиста рутину и защищающих указатели от некорректных действий. Таким образом, выражение `rs.Fields ("Article").value` означает значение поля "Article" текущего значения указателя на элемент базы данных (в нашем случае статей) и содержит текст статьи, которая соответствует текущей позиции указателя на все статьи. Переход к следующему элементу базы (смещение указателя) выполняется с помощью инструкции `Rs.MoveNext`. В приведенном выше примере это не делается, а попросту формируется ссылка на текст статьи в виде ее названия и отображается комментарий самой первой статьи, соответствующей результату запроса. Давайте попробуем отобразить все статьи нашей базы данных на главной странице в виде HTML. И еще, обратите особое внимание на директиву:

```
Link = "<a href= http://localhost/ArtTempl.asp?id=" _
        & rs.Fields("ID").value & ">" _
        & rs.Fields("Title").value & "</a>"
```

Она по сути формирует ссылку на файл, содержащий шаблон отображения самой статьи, причем передает идентификатор статьи ему в качестве параметра. Механизм работы файла шаблона может быть представлен следующим образом:

```
<%
TheID = Request.QueryString("id")
Set db = Server.CreateObject("ADODB.Connection")
db.Open "Articles"
sSQL = "SELECT * FROM Articles Where ID =" & TheID
Set rs = db.Execute(sSQL)
Response.Write Link & "<br>"
Response.Write rs.Fields("Title").value
Response.Write "<i><b> By " & rs.Fields("Author").value & "</b>"
Response.Write "<br>Subject: " & rs.Fields("Subject").value & "<br>"
Response.Write "Published: " & rs.Fields("Published").value & "</i><br>"
Response.Write rs.Fields("Article").value
db.Close
Set db = Nothing
%>
```

Первая строчка скрипта шаблона HTML присваивает переменной TheID значение, переданное ссылкой с использованием метода Request.QueryString. Далее открывается база данных, из которой читается статья (запись), соответствующая идентификатору, переданному из главного скрипта (index.asp).

Язык структурированных запросов — SQL

Настала пора разобраться с тем, что таится за строчками:

```
sSQL = "SELECT * FROM Articles"
Set rs = db.Execute(sSQL)
```

По сути, именно за этими двумя строчками кроется работа с нашей базой данных: первая представляет собой текстовую строку с запросом к базе; вторая — содержит директиву выполнения этого запроса с одновременным присвоением результата переменной (указателю на записи в базе данных). В рамках настоящей статьи мы не будем рассматривать SQL (Structured Query Language) во всех деталях, а остановимся лишь на тех его операторах, без понимания которых дальнейшая работа будет невозможна.

Для полноценной работы нам необходимо познакомиться с четырьмя операторами этого мощного языка, предназначенного специально для работы с базами данных.

```
DELETE FROM «Имя Таблицы» [WHERE Определение]
```

DELETE удаляет те ряды из «Имя Таблицы», которые удовлетворяют условию, определенному в «Определении», и возвращает число удаленных рядов. Если выполнить команду DELETE без

условия WHERE, то все ряды указанной таблицы будут удалены. В этом случае DELETE возвратит 0.

```
SELECT Выражение_Select,... [FROM ссылки_на_таблицы [WHERE where_определение]]
```

SELECT используется для извлечения рядов (записей) из одной или более таблиц. Выражение_Select определяет столбцы таблицы, значения которых необходимо извлечь. Выражение_Select можно заменить псевдонимом (alias) с помощью ключевого слова AS. Псевдоним используется в качестве идентификатора имени.
Примеры:

```
select concat AS Полное_Имя from Таблица ORDER BY Полное_Имя
```

INSERT используется для добавления новых записей в существующую таблицу. Допустимо две формы использования INSERT.

Форма 1:

```
INSERT [INTO] Имя_Таблицы [(имя_столбца,...)] VALUES (выражение,...),(...),...
```

Форма 2:

```
INSERT [INTO] Имя_Таблицы [(имя_столбца,...)] SELECT ...
```

Первая форма — INSERT ... VALUES — вставляет ряды на основании заданных значений.
Вторая форма — INSERT ... SELECT — вставляет ряды, выбранные из другой таблицы.
Примеры:

```
INSERT INTO Имя_Таблицы (Поле1,Поле2) VALUES(15,Поле1*2);
```

```
UPDATE Имя_Таблицы SET имя_поля1=выр1,имя_поля2=выр2,...[WHERE определение_where]
```

UPDATE обновляет поля существующей таблицы новыми значениями. Выражение SET показывает, какие поля (столбцы) должны быть изменены, и значения, которые должны быть им присвоены. Выражение WHERE, если оно есть, указывает, какие ряды должны быть обновлены. В противном случае операция применяется ко всем рядам таблицы.
Примеры:

```
Update WAPassword Set Password = 'passw' Where ID = 1
```

Обновляет значение поля Password в таблице WAPassword, записывая в поле, чей идентификатор ID равен 1 значение 'passw'.

Добавляем новую статью (UploadForm.asp и Upload2DBS.asp)

Теперь, когда мы разобрались с SQL, можно приступать к добавлению новой статьи, причем делать мы это будем прямо с сайта, а если быть точнее — непосредственно с HTML-формы. Для этого сначала создадим файл с самой формой и определим скрипт-реакцию на подтверждение (кнопку «Publish the article!»).

Прежде всего следует уточнить задачу на этом этапе. Итак, очевидно следующее:

- на загрузку статьи с сайта должен иметь право не каждый (следовательно, желательно предусмотреть пароль для доступа к этой функции);
- у каждой статьи есть определенная тема (рубрика), причем она не может быть произвольной, а должна выбираться из списка;
- список можно хранить непосредственно в HTML-файле и, каждый раз изменяя его, изменять сам файл. Это самый простой и быстрый способ;
- однако для того, чтобы позволить динамически изменять и пополнять этот список, рекомендуется держать его в базе данных. Это позволит пользователям произвольным образом изменять его содержимое и не потребует переделки формы.

Для простоты сначала рассмотрим вариант со встроенным («жестко прошитым») рубрикатором.

```
<html><head>
<title>Загрузка статьи</title>
</head> <body>
<FORM NAME="mainform" METHOD="GET" ENCTYPE="multipart/form-data"
ACTION="http://localhost/lr3/Upload2DBS.asp">
```

```
<b>Author: (*)</b><br>
<input type=TEXT size=56 name="Author"><br>
<!-- Поле имени автора статьи -->
<b>Title: (*)</b><br>
<input type="text" name="Title" size="56"><br>
<!-- Поле заголовка статьи -->
<b>Article Annotation:</b><br>
<textarea name="Annotation" cols="56"></textarea><br>
<!-- Поле аннотации к статье -->
<b>Article Text:</b><br>
<textarea cols="56" name="Article" rows="15"></textarea><br>
<!-- Поле текста самой статьи -->
<b>Type of the article: (*)</b><br>
<!-- Поле рубрикатора статьи -->

<select name="Subject">
<option>Economics</option>
<option>Education</option>
.......................
<option>Travel</option>
</select>

<br>
<b>Database access password: (*)</b><br>
<input type="password" name="Password">
<br>
<center>
<input type=SUBMIT value="Publish the Article!" name="SUBMIT">
<input type="reset" name="Reset" value="Reset Article Form">
</center>
</FORM>
</body>
</html>
```

Как видим, передача управления осуществляется благодаря директиве *ACTION="http://localhost/Upload2DBS.asp"*> в тэге формы. Тем самым указывается скрипт-ответ на реакцию пользователя после нажатия на кнопку «Publish the article!».

Теперь остановимся на селекторе рубрик. Как уже отмечалось, желательно перевести его содержимое в базу данных. Для этого выберем из нашей базы данных все встречающиеся там значения нашего рубрикатора и отсортируем полученный список в алфавитном порядке. После чего следует заменить тэг <select> в вышеприведенной форме на следующий вариант:

```
<select name="Subject">
<%
Set db = Server.CreateObject("ADODB.Connection")
db.Open "DSN=Articles"
SQLQuery = "SELECT distinct Subject FROM Articles ORDER BY Subject;"
Set rs = db.Execute(SQLQuery)
Do While NOT Rs.EOF
Response.Write "<option>"_
& rs.Fields("Subject").value & "</option>"
Rs.MoveNext
Loop
%>
</select>
```

... и все.

Что же должен делать наш скрипт-реакция?

Во-первых, следует позаботиться о том, чтобы все обязательные поля (а они отмечены звездочкой) были введены. Наиболее правильным способом проверки этого является скрипт. Для этого достаточно добавить в определение тэга формы параметр onSubmit="preprocess();". Здесь как нельзя кстати видно преимущество языков описания сценариев (JavaScript, Jscript, VBScript) перед ASP. ASP выполняется на стороне сервера, а перегружать связь «клиент-сервер» простой

проверкой типа «введены ли значения», согласитесь, неправильно. Однако специально в целях обучения мы будем делать это с помощью ASP.

И еще. Прежде чем приступить к выполнению нашей задачи, следует сказать еще об одной небольшой проблеме: когда мы будем вставлять в ячейку базы данных текст, его форматирование (даже самое простое) не сохранится, и в дальнейшем его невозможно будет корректно отобразить на HTML-странице, поскольку поле Memo хранит неформатированную строку текста. Чтобы избежать этого, следует написать функцию, которая позволит производить примитивное HTML-форматирование введенного текста статьи, перед тем как записать его в базу данных. Другими словами, в самом примитивном случае (хотя бы для того, чтобы сохранить исходную разбивку на строки) функция должна вставлять символы конца строки там, где во входном потоке имеются символы переноса строк.

```
<html>
<body>

<%
Set db = Server.CreateObject("ADODB.Connection")
db.Open "DSN=Articles"
SQLQuery = "Select * From WAPassword Where ID = 1"
Set rs = db.Execute(SQLQuery)
DBP = rs.Fields("Password").value
db.Close
Set db = Nothing

Function FormatStr(InString)
on Error resume next
InString = Replace(InString, CHR(13) & CHR(10), "<br>")
InString = Replace(InString, CHR(10) & CHR(10), "</P><P>")
InString = Replace(InString, CHR(10), "<BR>")
InString = Replace(InString, "'", "`")
InString = Replace(InString, CHR(34), "/")
FormatStr = InString
End Function

ErrA = 0
ErrT = 0
ErrP = 0
ErrC = 0

AUT = Request.QueryString ("Author")
If AUT = "" Then ErrA = 1 End If
AUT = FormatStr(AUT)

TIT = Request.QueryString ("Title")
If TIT = "" Then ErrT = 1 End If
TIT = FormatStr(TIT)

ART = Request.QueryString ("Article")
If ART = "" Then
  ART = "No Text"
  If File1 = "No" Then ErrC = 1 End If
End If

ART = FormatStr(ART)

SBJ = Request.QueryString ("Subject")
'IsTopNew = Upload.Form("IsTopNewSelector")

ANN = Request.QueryString ("Annotation")
ANN = FormatStr(ANN)
If ANN = "" Then
  ANN = "NA"
```

```
End If

Password = Request.QueryString ("Password")
If Password = DBP Then
 If ErrA = 0 and ErrT = 0 Then
  Set db = Server.CreateObject("ADODB.Connection")
  db.Open "DSN=Articles"
  sSQL="insert into Articles (Author,Title,Article,Subject,Published,Annotation) "_
& "values('" & AUT & "', '" & TIT & "','" & ART & "','" & SBJ & "','" & TIM &"', '" & ANN & "')"
  Set rs = db.Execute(sSQL)
  db.Close
  Set db = Nothing
 End If
Else
 ErrP = 1
End If

N = Now()
If ErrP = 0 and ErrA = 0 and ErrT = 0 and ErrC = 0 Then
  Response.Write "Article uploaded successfully at " & N &"<br>"
End If

If ErrP = 1 Then
  Response.Write "Error! Article database access " & "denied! Incorrect password!"
End If

If ErrA = 1 Then
  Response.Write "<br>" & "Error! No Author specified!"
End If

If ErrT = 1 Then
  Response.Write "<br>" & "Error! No Title specified!"
End If

If ErrC = 1 Then
  Response.Write "<br>" & "Error! You should " & "specify article text!"
End If
%>
</body>
</html>
```