Глава 15. Язык Perl и CGI-программирование

Учебник по Perl

Содержание [скрыть]

Основные понятия

Основу "всемирной паутины" WWW составляют *Web-узлы.* Это компьютеры, на которых выполняется специальная программа – *Web-сервер,* ожидающая запроса со стороны клиента на выдачу документа. Документы сохраняются на Web-узле, как правило, в формате HTML. Клиентом Web-сервера является программа-браузер, выполняющаяся на удаленном компьютере, которая осуществляет запрос к Web-серверу, принимает запрошенный документ и отображает его на экране.

Аббревиатура CGI *(Common Gateway Interface)* обозначает часть Web-сервера, которая может взаимодействовать с другими программами, выполняющимися на этом же Web-узле. В этом смысле она является шлюзом *(gateway* – шлюз) для передачи данных, полученных от клиента, программам обработки – таким, как СУБД, электронные таблицы и др. CGI включает общую среду (набор переменных) и протоколы для взаимодействия с этими программами.

Общая схема работы CGI состоит из следующих элементов.

1. Получение Web-сервером информации от клиента-браузера. Для передачи данных Web-серверу в языке HTML имеется средство, называемое *форма.* Форма задается в HTML-документе при помощи тэгов <FORM>. . .</FORM> и состоит из набора *полей ввода,* отображаемых браузером в виде графических

элементов управления: селекторных кнопок, опций, строк ввода текста, управляющих кнопок и т. д. (рис. 15.1).

2. Анализ и обработка полученной информации. Данные, извлеченные из HTML-формы, передаются для обработки CGI-программе. Они не всегда могут быть обработаны CGI-программой самостоятельно. Например, они могут содержать запрос к некоторой базе данных, которую CGI-программа читать "не умеет". В этом случае CGI-программа на основании полученной информации формирует запрос к компетентной программе, выполняющейся на том же компьютере. CGI-программа может быть написана на любом языке программирования, имеющем средства обмена данными между программами. В среде UNIX для этой цели наиболее часто используется язык Perl. Так как UNIX является наиболее популярной операционной системой для Web-серверов, то можно считать Perl наиболее популярным языком CGI-программирования. Программа на языке Perl представляет собой последовательность операторов, которые *интерпретатор* языка выполняет при каждом запуске без преобразования исходного текста программы в выполняемый двоичный код. По этой причине CGI-программы называют также *CGI-сценариями* или *CGI-скрип-тами.*

3. Создание нового HTML-документа и пересылка его браузеру.

После обработки полученной информации CGI-программа создает динамический или, как говорят, виртуальный HTML-документ, или формирует ссылку на уже существующий документ и передает результат браузеру.

Рис 15.1. Пример отображения HTML-формы браузером

HTML-формы

HTML-формы предназначены для пересылки данных от удаленного пользователя к Web-серверу. С их помощью можно организовать простейший диалог между пользователем и сервером (например, регистрацию пользователя на сервере или выбор нужного документа из представленного списка). Формы поддерживаются всеми популярными браузерами.

(Различные аспекты передачи данных Web-серверу будут рассмотрены в разделе 15.3.)

Мы предполагаем, что читатель знаком с основами языка HTML и структурой HTML-документа. В этом разделе рассмотрены средства HTML, используемые для создания форм. В описании тэгов приведены только наиболее употребительные атрибуты и опущены атрибуты, специфические для отдельных браузеров.

Тэг *<FORM>*

```
<FORM ACTION="URL" METHOO=метод_перела чи ЕНСТУРЕ=М1МЕ-!гип> \
солержание_формы
</FORM>
```

В HTML-документе для задания формы используются тэги <FORM>. . .</FORM>, отмечающие, соответственно, начало и конец формы. Документ может содержать несколько форм, но они не могут быть вложены одна в другую. Тэг <FORM> имеет атрибуты ACTION, METHOD и ENCTYPE. Отдельные браузеры (Netscape, Internet Explorer) поддерживают дополнительные атрибуты помимо стандартных, например, CLASS, NAME, STYLE и др.

Атрибут ACTION – единственный обязательный. Его значением является адрес (URL) CGI-программы, которая будет обрабатывать информацию, извлеченную из данной формы.

Атрибут METHOD определяет метод пересылки данйых, содержащихся в форме, от браузера к Web-серверу. Он может принимать два значения; GET (по умолчанию) и POST.

Замечание

Взаимодействие между клиентом-браузером и Web-сервером осуществляется по правилам, заданным протоколом HTTP, и состоит из *запросов* клиента и ответов сервера. Запрос разбивается на три части. В первой строке запроса содержится HTTP-команда, называемая *методом,* URL запрашиваемого файла и номер версии протокола HTTP. Вторая часть– заголовок запроса. Третья часть – тело запроса, собственно данные, посылаемые серверу. *Метод* сообщает серверу о цели запроса. В протоколе HTTP определены несколько методов. Для передачи данных формы в CGI-программу используются два метода: GET И POST.

При использовании метода GET данные формы пересылаются в составе URL запроса, к которому присоединяются после символа "?" в виде совокупности пар *переменная=значение,* разделенных символом "&". В этом случае первая строка запроса может иметь следующий вид:

После выделения данных из URL сервер присваивает их переменной среды QUERY_STRING, которая может быть использована CGI-программой.

При использовании метода POST данные формы пересылаются Web-серверу в теле запроса, после чего передаются сервером в CGI-программу через стандартный ввод.

Значением атрибута ENCTYPE является *медиа-тип,* определяющий формат кодирования данных при передаче их от браузера к серверу. Браузер кодирует данные, чтобы избежать их искажения в процессе передачи. Возможны два значения этого атрибута: *appiication/x-wuw-form-uriencoded,* используемое ПО умолчанию, И *multipart/form-data.*

Замечание

Одним из первых применений Internet была электронная почта, ориентированная на пересылку текстовых сообщений. Часто возникает необходимость вместе с текстом переслать данные в нетекстовом формате, например, упакованный zip-файл, рисунок в формате GIF, JPEG и т. д. Для того чтобы пересылать средствами электронной почты такие файлы без искажения, они кодируются в соответствии с некоторым стандартом. Стандарт MIME *(Multipurpose Internet Mail Extensions,* многоцелевые расширения электронной почты для Internet) определяет набор *MIME-типов,* соответствующих различным типам данных, и правила их пересылки по электронной почте. Для обозначения MIME-типа используется запись вида *тип/подтип. Тип* определяет общий тип данных, например, *text, image, application* (тип *application* обозначает специфический внутренний формат данных, используемый некоторой программой), а *подтип* – конкретный формат внутри типа данных, например, *application/zip, image/gif, text/html.* MIME-типы нашли применение в Web, где они называются также *ме-dua-типами,* для идентификации формата документов, передаваемых по протоколу HTTP. В HTML-форме атрибут ENCTYPE определяет медиа-тип, который используется для кодирования и пересылки специального типа данных – содержимого формы.

Как видно из примера на рис. 15.1, форма отображается в окне браузера в виде набора стандартных элементов управления, используемых для заполнения полей формы значениями, которые затем передаются Web-серверу. Значение вводится в поле ввода пользователем или назначается по умолчанию. Для создания полей средствами языка HTML существуют специальные тэги: <INPUT>, <SELECT>, <TEXTAREA>, которые могут употребляться только внутри тэга <FORM>.

Тэг *<INPUT>*

```
<INPUT TYPE=тип_лоля_ввода ЫАМЕ=имя_лоля_ввода друте_атрибуты>
```

Наиболее употребительный тэг, с помощью которого можно генерировать внутри формы поля для ввода строки текста, пароля, имени файла, различные кнопки. Он имеет два обязательных атрибута: TYPE и NAME. Атрибут TYPE определяет тип поля: селекторная кнопка, кнопка передачи и др. Атрибут NAME определяет имя, присваиваемое полю. Оно не отображается браузером, а используется в качестве идентификатора значения, передаваемого Web-серверу. Остальные атрибуты меняются в зависимости от тира поля. Ниже приведено описание типов полей, создаваемых при помощи тэга <INPUT>, и порождаемых ими элементов ввода.

- TYPE=TEXT. Создает элемент для ввода строки текста. Дополнительные атрибуты:

- MAXLENGTH=n Задает максимальное количество символов, разрешенных в текстовом поле. По умолчанию не ограничено,
- SIZE=n. Максимальное количество отображаемых символов.
- TYPE=начальное_значение. Первоначальное значение текстового поля.

```
TYPE=PASSWORD
```

Создает элемент ввода строки текста, отличающийся от предыдущего только тем, что все вводимые символы представляются в виде символа *.

> **Замечание**
>
> Поле PASSWORD не обеспечивает безопасности введенного текста, так KaKjna сервер он передается в незашифрованном виде. /-"'"

```
TYPE=FILE
```

Создает поле для ввода имени локального файла, сопровождаемое кнопкой **Browse.** Выбранный файл присоединяется к содержимому формы при пересылке на сервер. Имя файла можно ввести непосредственно, или, воспользовавшись кнопкой **Browse,** выбрать его из диалогового окна, отображающего список локальных файлов. Для корректной передачи присоединенного файла следует установить значения атрибутов формы равными ENCTYPE="multipart/form-data" И METHOD=POST. В Противном случае будет передана введенная строка, то есть маршрутное имя файла, а не его содержимое. Дополнительные атрибуты MAXLENGTH и SIZE имеют тот же смысл, что и для элементов типа TEXT и PASSWORD.

- TYPE=CHECKBOX Создает элемент-переключатель, принимающий всего два значения (on/off, вкл./выкл., истина/ложь) и отображаемый в виде квадратной кнопки. Элементы-переключатели CHECKBOX можно объединить в группу, установив одинаковое значение атрибута NAME для всех ее элементов. Дополнительные атрибуты:
- TYPE=строка Значение, которое будет передано серверу, если данная кнопка выбрана. Если кнопка не выбрана, значение не передается. Обязательный атрибут.
- CHECKED Если указан атрибут CHECKED, элемент является выбранным по умолчанию.

Если переключатели образуют группу, то передаваемым значением является строка разделенных запятыми значений атрибута VALUE всех выбранных элементов.

- TYPE=RADIO Создает элемент "радиокнопка", существующий только в составе группы подобных элементов, из которых может быть выбран только один. Все элементы группы должны иметь одинаковое значение атрибута NAME. Отображается в виде круглой кнопки. Дополнительные атрибуты:

- VALUE=Строка Обязательный атрибут, значение которого передается серверу при выборе данной кнопки. Должен иметь уникальное значение для каждого члена группы.
- CHECKED Устанавливает элемент выбранным по умолчанию. Один и только один элемент в группе должен иметь этот атрибут.
- TYPE=SUBMIT Создает кнопку передачи, нажатие которой вызывает пересылку на сервер всего содержимого формы. По умолчанию отображается в виде прямоугольной кнопки с надписью **Submit.** Дополнительный атрибут
- TYPE=яазвание_кнолки позволяет изменить надпись на кнопке. Атрибут NAME для данного элемента может быть опущен. В этом случае значение кнопки не включается в список параметров формы и не передается на сервер.

Если атрибуты NAME И VALUE Присутствуют, Например,

```
<INPUT TYPE=SUBMIT NAME="submit_button" VALUE="OK">,
```

то в список параметров формы, передаваемых на сервер, включается параметр submit_button="OK". Внутри формы могут существовать несколько кнопок передачи.

- TYPE=RESET. Создает кнопку сброса, нажатие которой отменяет все сделанные изменения, восстанавливая значения полей формы на тот момент, когда она была загружена. По умолчанию отображается в виде прямоугольной кнопки с надписью **Reset.** Надпись можно изменить при помощи дополнительного атрибута
- *TYPE.=название_кнопки.* Значение кнопки **Reset** никогда не пересылается на сервер, поэтому у нее отсутствует атрибут NAME.
- TYPE=IMAGE. Создает элемент в виде графического изображения, действующий аналогично кнопке **Submit.**

Дополнительные атрибуты:

- *TYPE=тип_изображения.* Задает ссылку (url) на файл с графическим изображением элемента.
- *TYPE=тип_въ1равнивания.* Задает тип выравнивания изображения относительно текущей строки текста точно так же, как одноименный атрибут тэга <IMG>.

Если на изображении элемента щелкнуть мышью, то координаты указателя мыши в виде NAME.x=n&NAME.y=m включаются браузером в список параметров формы, посылаемых на сервер.

- TYPE=HIDDEN Создает скрытый элемент, не отображаемый пользователю. Информация, хранящаяся в скрытом поле, всегда пересылается на сервер и не может быть изменена ни пользователем, ни браузером. Скрытое поле можно использовать, например, в следующем случае. Пользователь заполняет форму и отправляет ее серверу. Сервер посылает пользователю для заполнения вторую форму, которая частично использует информацию, содержащуюся в первой форме. Сервер не хранит историю диалога с пользователем. Он обрабатывает каждый запрос независимо и при получении

второй формы не будет знать, как она связана с первой. Чтобы повторно не вводить уже введенную информацию, можно заставить CGI-програм-му, обрабатывающую первую форму, переносить необходимые данные в скрытые поля второй формы. Они не будут видимы пользователем и, в то же время, доступны серверу. Значение скрытого поля определяется атрибутом VALUE.

Тэг *<SELECT>*

```
<SELECT NAME=HM*_n<afl# SIZE=n MULTIPLE>
элементы OPTION </SELECT>
```

Тэг <SELECT> предназначен для того, чтобы организовать внутри формы выбор из нескольких вариантов без применения элементов ввода типа CHECKBOX и RADIO. Дело в том, что если элементов выбора много, то представление их в виде переключателей и радиокнопок увеличивает размеры формы, делая ее труднообозримой. С помощью тэга <SELECT> варианты выбора более компактно представляются в окне браузера в виде элементов ниспадающего меню или списка прокрутки. Тэг имеет следующие атрибуты.

- NAME=*строка* Обязательный атрибут. При выборе одного или нескольких элементов формируется список выбранных значений, который передается на сервер под именем NAME.
- SIZE=n Устанавливает число одновременно видимых элементов выбора. Если n=1, то отображается ниспадающее менй, если n>1, то список прокрутки с n одновременно видимыми элементами.
- MULTIPLE Означает, что из меню или списка можно выбрать одновременно несколько элементов. Если этот атрибут задан, то список выбора ведет себя как группа переключателей CHECKBOX, если не задан – как группа радиокнопок RADIO.

Элементы меню задаются внутри тэга <SELECT> при помощи тэга <OPTION>:

```
<OPTION SELECTED TYPE=строка>содержимое_тэг'а</OPTION>
```

Закрывающий тэг </OPTION> не используется. Атрибут VALUE содержит значение, которое пересылается серверу, если данный элемент выбран из меню или списка. Если значение этого атрибута не задано, то по умолчанию оно устанавливается равным содержимому тэга <OPTION>. Например, элементы

```
<OPTION VALUE=Red>Red <OPTION>Red
```

имеют одно значение Red. В первом случае оно установлено явно при помощи атрибута VALUE, во втором – по умолчанию. Атрибут SELECTED изначально отображает элемент как выбранный.

Тэг *<TEXTAREA>*

```
<TEXTAREA HAME=имя ROWS=m COLS=n>
```

текст

```
</TEXTAREA>
```

Создает внутри формы поле для ввода многострочного текста, отображаемое в окне браузера в виде прямоугольной области с горизонтальной и вертикальной полосами прокрутки. Для пересылки на сервер каждая введенная строка дополняется символами %OD%OA (ASCII-символы "Возврат каретки" и "Перевод строки" с предшествующим символом *%),* полученные строки объединяются в одну строку, которая и отправляется на сервер под именем, задаваемым атрибутом NAME. Атрибуты:

- NAME ' __ Необходимый атрибут, используемый для идентификации данных при пересылке на сервер.
- COLS=n Задает число столбцов видимого текста.
- ROWS=n Задает число строк видимого текста.

Между тэгами <textarea> и </textarea> можно поместить текст, который будет отображаться по умолчанию.

Пример формы

Ниже представлен пример формы, включающей набор характерных полей и HTML-код, использованный для ее создания.

```
<htmlxheadxtitle>npHMep формы</^д^1ex/пеа3>
<body>
<b2>Регистрационная страница Клуба любителей фантастики</b2>
Заполнив анкету, вы сможете пользоваться нашей электронной библиотекой.
<br>
<form method="get" action="/cgi-bin/registrar.cgi">
<pre>
Введите регистрационное имя: <input type="text" name="regnarae">
Введите пароль: <input type="password" name="passwordl" max-length=8>
Подтвердите пароль: <input type="password" name="password2" max-length=8>
</pre>
Ваш возраст:
<input type="radio" name="age" value="lt20" checked >До 20
<input type="radio" name="age" value="20_30">20-30
<input type="radio" name="age" value="30_50">30-50
<input type="radio" name="age" value="gt50">CTapiue 50
<brxbr>
На каких языках читаете:
<input type="checkbox" name="language" value="russian" checked>pycciorii
<input type="checkbox" name="language" ^a1ие="епд!1зЬ">английский
<input type="checkbox" name="language" Уа1ие="£гепсЬ">французский
<input type="checkbox" name="language" value="gennan">HeMeuKHH
```

```
<brxbr>
Какой формат данных является для Вас предпочтительным
<br>
<select name="format" size=2 >
<option selected value="HTML">HTML
<option value="Plain text">Plain text
<option value="PostScript">PostScript
<option value="PDF">PDF </select> <brxbr>
Ваши любимые авторы: <br> <textarea name="wish" cols=40 rows=3>
</textarea> <brxbr>
<input type="submit" value="OK"> <input type="reset" ya!ue="0тменить">
</form>
</body> \
</html>
```

Данная форма содержит:

- текстовое поле для ввода регистрационного имени пользователя;
- текстовое поле для ввода пароля, отображаемого в окне символами *;
- текстовое поле для подтверждения пароля, также отображаемого символами *;
- группу радиокнопок для указания возраста пользователя (единственный выбор);
- группу переключателей для указания языков, которыми владеет пользователь (множественный выбор);
- список прокрутки для указания предпочтительного формата данных (выбор из ограниченного списка);
- блок ввода многострочного текста для перечисления любимых авторов (неизвестное заранее количество строк);
- кнопку передачи с меткой **OK** (у этого элемента отсутствует атрибут NAME, он не нужен, так как в данном примере всего одна кнопка передачи, а, значит, CGI-программе нет необходимости определять, от какой именно кнопки поступила команда передачи данных);
- кнопку сброса с меткой **Отменить.**

Итак, пользователь заполнил форму и щелкнул кнопку передачи **Submit.** Дальнейшее прохождение данных выглядит следующим образом.

- Информация кодируется и пересылается на Web-сервер, который передает ее для обработки CGI-программе.
- CGI-программа обрабатывает полученные данные, возможно, обращаясь за помощью к другим программам, выполняющимся на том же компьютере, и генерирует новый "виртуальный" HTML-документ, либо определяет ссылку на уже имеющийся.

- Новый HTML-документ или ссылка передаются CGI-Программой Web-серверу для возврата клиенту.

Рассмотрим эти шаги более подробно.

Передача информации CGI-программе

Как мы уже знаем, существуют два метода кодирования информации, содержащейся в форме: стандартный метод *application/x-www-form-urlencoded,* используемый по умолчанию, и дополнительный *multipart/form-data.* Второй метод нужен только в том случае, если к содержимому формы присоединяется локальный файл, выбранный при помощи элемента формы < INPUT TYPE=FILE>. В остальных случаях следует использовать метод кодирования по умолчанию.

Схема кодирования *application/x-www-form-urlencoded* одинакова для обоих методов пересылки GET и POST и заключается в следующем.

Для каждого элемента формы, имеющего имя, заданное атрибутом МИМЕ, формируется пара *"name=value",* где *value* — значение элемента, введенное пользователем или назначенное по умолчанию. Если значение отсутствует, соответствующая пара имеет вид *"пате—'.* Для радиокнопок и переключателей используются значения только выбранных элементов. Если элемент выбран, а значение атрибута VALUE не определено, по умолчанию используется значение "ON".

Все пары объединяются в строку, в качестве разделителя служит символ &. Так как имена и значения представляют собой обычный текст, то они могут содержать символы, недопустимые в составе URL (метод GET пересылает данные как часть URL). Такие символы заменяются последовательностью, состоящей из символа % и их шестнадцатеричного ASCII-кода. Символ пробела может заменяться не только кодом %20, но и знаком + (плюс). Признак конца строки, встречающийся в поле TEXTAREA, заменяется кодом %OD%OA. Такое кодирование называется URL-кодированием.

Закодированная информация пересылается серверу одним из методов GET или POST. Основное отличие заключается в том, как метод передает информацию CGI-программе.

При использовании метода GET данные формы пересылаются серверу в составе URL запроса, к которому добавляются после символа ? (вспомним, что запрос — это формализованный способ обращения браузера к Web-серверу). Тело запроса в этом случае является пустым. Для формы из примера 15.1 запрос выглядит следующим образом:

```
GET /cgi-bin/registrar.cgi? regname = bobSpasswordl = rumataSpassword2 = rumata&age =
lt20&language = russian&format = HTML&wish = %F6%C5%CC%Dl%DA%CE%D9 HTTP/1.0
```

(заголовки запроса, сообщающие серверу, информацию о клиенте) <пусто> (тело запроса)

Часть URL после символа "?" называется *строкой запроса.* Web-сервер, получив запрос, присвоит переменной среды QUERY_STRING значение строки запроса и вызовет CGI-программу, обозначенную в

первой части URL до символа "?": /cgi-bin/registrar.cgi. **CGI-Программа** registrar.cgi сможет затем обратиться к переменной QUERY_STRING для обработки закодированных в ней данных.

Обратите внимание на то, что данные, введенные в поле типа PASSWORD, передаются открытым текстом без шифрования. При передаче данных методом GET они в составе URL помещаются в файл регистрации доступа access.log, обычно открытый для чтения всем пользователям. Таким образом "секретные" данные, введенные в поле типа PASSWORD, оказываются доступными посторонним.

> **Замечание**
>
> Метод GET позволяет передавать данные CGI-программе вообще без использования форм. Информацию, содержащуюся в приведенном выше URL, можно передать при помощи следующей гиперссылки, помещенной в HTML-документ: <A HREF="http://www.domain/cgi-bin/registrar.cgi?regname= 4>bobspasswordl=rumata&password2=rimata&age=lt20slanguage=russian& 4>format=HTML&wish=%F6%C5%CC%Dl%DA%CE%D9 ">CC1-программа</A>, заменив в этом фрагменте символ "&" его символьным примитивом &#38 или &amp для правильной интерпретации браузером.

К сожалению, эта информацдагявдяется статической. Форма же позволяет менять данные.

Строка запроса — не единственный способ передачи данных через URL. Другим способом является *дополнительная информация о пути (extra path information),* представляющая собой часть URL, расположенную после имени CGt-программы. Сервер выделяет эту часть и сохраняет ее в переменной среды PATH_INFO. CGI-программа может затем использовать эту переменную для извлечения данных. Например, URL

```
http://www.domain/cgi-bin/registrar.cgi/4>regname = bobspasswordl
= rumataspassword2 = r\amata&age = lt20&language
= 4>russian&format = HTML&wish = %F6%C5%CC%Dl%DA%CE%D9
```

содержит уже знакомые нам данные, но не в виде *строки запроса, а* в виде *дополнительной информации о пути.* При получении запроса с таким URL сервер сохранит данные в переменной среды

```
PATH_INFO = /regname = bobspasswordl = rumataspassword2 = rumata&age = lblt20&language =
russian&format = HTML&wish = %F6%C5%CC%Dl%DA%CE%D9"
```

Название объясняется тем, что обычно этим способом передается информация о местоположении какого-либо файла *(extra path information).* Например, URL

```
http://www.domain/cgi-bin/registrar.cgi/texts/jdk_doc.txt
```

содержит дополнительную информацию PATH_iNFO=/texts/jdk_doc.txt" о местонахождении файла jdk_doc.txt относительно корневого каталога дерева документов. Другая переменная среды

PATHJTRANSLATED содержит информацию об абсолютном местоположении файла в файловой системе, например,

```
PATH_TRANSLATED="/hcme/httpd/docs/texts/jdk_doc.txt"
```

а переменная DOCUMENT_ROOT содержит путь к корневому каталогу дерева документов, В нашем случае DOCUMENT_ROOT="/home/httpd/docs/".

При использовании метода POST данные формы пересылаются серверу в теле запроса. Если в примере 15.1 вместо метода GET использовать метод POST

```
<form method="post" action="/cgi-bin/registrar.cgi">,
```

то запрос клиента будет иметь следующий вид:

```
POST /cgi-bin/registrar.cgi HTTP/1.1
```

(заголовки запроса, сообщающие серверу информацию о клиенте)

```
Content-length: 126
regname=bob&passwordl=rumata&password2=ruinata&age=lt20&language=russian&
4>format=HTML&wish=%F6%C5%CC%Dl%DA%CE%D9
```

В этом фрагменте среди прочих заголовков выделен заголовок content-length, сообщающий серверу количество байт, переданных в теле запроса. Это значение сервер присваивает переменной среды CONTENT_LENGTH, а данные посылает в стандартный ввод CGI-программы.

Методы GET и POST имеют свои достоинства и недостатки. Метод GET обеспечивает лучшую производительность при пересылке форм, состоящих из небольшого набора коротких полей. При пересылке большого объема данных следует использовать метод POST, так как браузер или сервер могут накладывать ограничения на размер данных, передаваемых в составе URL, и отбрасывать часть данных, выходящую за границу. Метод POST, к тому же, является более надежным при пересылке конфиденциальной информации.

CGL-сценарии

Назначение CGI-программы — создать новый HTML-документ, используя данные, содержащиеся в запросе, и передать его обратно клиенту. Если такой документ уже существует, то передать ссылку на него. Какой язык можно использовать для написания CGI-программ? Сам интерфейс CGI не накладывает ограничений на выбор языка программирования. Зная, какую задачу решает CGl-программа и каким образом она получает входную информацию, мы можем назвать свойства, которыми должен обладать язык CGI-программирования.

- Средства обработки текста. Необходимы для декодирования входной информации, поступающей в виде строки, состоящей из отдельных полей, разделенных символами-ограничителями.

- Средства доступа к переменным среды. Необходимы, так как с помощью переменных среды данные передаются на вход CG.I-программы.
- Возможность взаимодействовать с другими программами. Необходима для обращения к СУБД, программам обработки графики и другим специальным программам.

Выбор языка зависит и от операционной системы Web-сервера. Большая часть имеющихся серверов предназначена для работы под управлением операционной системы UNIX. Учитывая эти соображения, мы можем заключить, что язык Perl, обладающий развитыми средствами обработки текста и создания сценариев, первоначально созданный для работы в ОС UNIX и перенесенный на множество других платформ, является наиболее подходящим средством создания сценариев CGI. Кроме того, CGI-програм-мирование на языке Perl имеет поддержку в виде готовых модулей CPAN, свободно доступных в сети Internet.

CGI-сценарий на языке Perl – это программа, имеющая свою специфику. Она, как правило, генерирует HTML-документ, посылаемый клиенту в виде *ответа сервера.* Ответ сервера, так же, как и запрос клиента, имеет определенную структуру. Он состоит из ^следующих трех частей:

1. Строка состояния, содержащая три поля: номер версии протокола HTTP, код состояния и краткое описание состояния, например:

```
HTTP/1.0 200 OK f запрос клиента обработан успешно
HTTP/1.0 404 Not Found # Документ по указанному адресу не существует
```

2. Заголовки ответа, содержащие информацию о сервере и о возвращаемом HTML-документе, например:

```
Date: Mon, 26 Jul 1999 18:37:07 GMT # Текущая дата и время
Server: Apache/1.3.6 ::,, ·.# Имя и номер версии сервера
Content-type: text/html tt Описывает медиа-тип содержимого
```

3. Содержимое ответа – HTML-документ, являющийся результатом выполнения CGI-программы.

CGI-программа передает результат своей работы (HTML-документ) серверу, который возвращает его клиенту. При этом сервер не анализирует и не изменяет полученные данные, он может только дополнять их некоторыми заголовками, содержащими общую информацию (например, текущая дата и время) и информацию о самом себе (например, имя и версия сервера). Информация о содержимом ответа формируется CGI-программой и должна содержать как минимум один заголовок, сообщающий браузеру формат возвращаемых данных:

```
Content-type: text/html
```

Замечание

Информацию о заголовках можно найти в спецификации протокола HTTP. Мы же ограничимся еще одним примером. Если в качестве ответа клиенту посылается статический документ, например, подтверждение о получении заполненной формы, то неэффективно каждый раз создавать его заново. Лучше создать один раз и сохранить в файле. В этом случае CGI-сценарий вместо заголовка Content-type: media-type, описывающего формат данных, формирует заголовок Location: URL, указывающий серверу местонахождение документа, который следует передать клиенту.

Заголовки отделяются от содержимого документа пустой строкой.

Напишем простейший CGI-сценарий, посылающий пользователю HTML-страницу с приветствием

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
print "<html><head><title>HELLO</titlex/head>\n";
print "<body>\n";
print "<h2>Вас приветствует издательство BXB - Санкт-Петербург</п2>\n"; print "
</bodyx/html>\n";
```

Если поместить файл hello.cgi в каталог CGI-программ Web-сервера, а затем обратиться к нему из браузера, то браузер отобразит HTML-документ, созданный программой hello.cgi (рис. 15.2).

Замечание

Большинство Web-серверов по умолчанию предполагают, что файлы GGi-сце-нариев находятся в специальном каталоге, обычно называемом cgi-bin. Можно настроить сервер таким образом, чтобы все файлы, находящиеся в определенном каталоге, он воспринимал не как обычные документы, а как выполняемые сценарии. Можно также указать серверу, что все файлы с определенным расширением (например, .cgi) должны рассматриваться как CGI-сценарий. Когда пользователь открывает URL, ассоциированный с CGI-программой. клиент посылает запрос серверу, запрашивая файл. Сервер распознает, что запрошенный адрес является адресом CGI-программы, и пытается выполнить эту программу. Подробности конфигурирования Web-серверов можно найти в соответствующей литературе и документации на конкретный сервер.

Рис 15.2. Web-страница, сформированная программой hello.cgi

Переменные среды CGI

В зависимости от метода данные формы передаются в CGI-программу или через стандартный ввод (POST), или через переменную среды QUERY_STRING (GET). Помимо этих данных CGI-программе доступна и другая информация, поступившая от клиента в заголовках запроса или предоставленная Web-сервером. Эта информация сохраняется в переменных среды UNIX. С некоторыми из них мы уже познакомились ранее. В табл. 15.1 перечислены переменные, обычно используемые в CGI.

Таблица 15.1. *Переменные среды CGI*

Переменная среды	Описание
GATEWAYJNTERFACE	Версия CGI, которую использует сервер
SERVER_NAME	Доменное имя или IP-адрес сервера
SERVER_SOFTWARE	Имя и версия программы-сервера, отвечающей на запрос клиента (например, Apache 1 .3)
SERVER_PROTOCOL	Имя и версия информационного протокола, который был использован для запроса (например, HTTP 1 .0)
SERVER_PORT	Номер порта компьютера, на котором работает сервер (по умолчанию 80)
REQUEST_METHOD	Метод, использованный для выдачи запроса (GET, POST)
PATHJNFO	Дополнительная информация о пути
PATHJRANSLATED	Та же информация, что и в переменной PATHJNFO с префиксом, задающим путь к корневому каталогу дерева Web-документов
SCRIPT_NAME	Относительное маршрутное имя CGI-сценария (например, /cgi-bin/program.pl)
DOCUMENT_ROOT	Корневой каталог дерева Web-документов
QUERY_STRING	Строка запроса– информация, переданная в составе URi запроса после символа "?"

REMOTE_HOST	Имя удаленной машины, с которой сделан запрос
REMOTE_ADDR	IP-адрес удаленной машины, с которой сделан запрос
REMOTE_USER	Идентификационное имя пользователя, посылающего запрос
CONTENT_TYPE	Медиа-тип данных запроса, например, "text/html".
CONTENT_LENGTH	Количество байт в теле запроса, переданных в CGI-программу через стандартный ввод
HTTP_HOST	Хост-имя компьютера, на котором работает сервер
HTTP_FROM	Адрес электронной почты пользователя, направившего запрос
HTTP_ACCEPT	Список медиа-типов, которые может принимать клиент
HTTP_USER_AQENT	Браузер, которым клиент пользуется для выдачи запроса
HTTP_REFERER	URL документа, на который клиент указывал перед обращением к CGI-программе

Имена переменных среды CGI на разных Web-серверах могут различаться. Следует обратиться к документации на соответствующий сервер.

CGI-программа на языке Perl имеет доступ к переменным среды через специальный предопределенный хеш-массив %ENV, к элементам которого можно обратиться по ключу, совпадающему с именем переменной среды. Ниже приведены пример CGI-сценария, формирующего HTML-документ с информацией о всех установленных переменных среды, и отображение этого документа в окне браузера.

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
print "<html>\n";
print "<head><title>IlepeMeHHbie cpe№</titlex/head>\n";
print "<boду><b2>Переменные среды</b2>\n";
print "<hrxpre>\n";
foreach $name (sort(keys %ENV))
{
print "$name: $ENV($name}\n";
```

```
}
print "<hrx/pre>\n";
print "</bodyx/html>\n";
```

Рис 15.3. Информация о переменных среды CGI

Обработка данных формы

Данные формы поступают в CGI-программу в закодированном виде, поэтому в качестве первого шага CGI-сценарий должен выполнить декодирование полученной информации. При пересылке данных методом GET данные формы присваиваются переменной среды QUERY_STRING, при передаче методом POST – передаются в программу через стандартный ввод и тоже могут быть присвоены некоторой внутренней переменной. Таким образом, декодирование данных сводится к следующей последовательности манипуляций со строкой:

- замена каждой группы %hh, состоящей из шестнадцатеричного ASCII-кода hh с префиксом %, на соответствующий ASCII-символ;
- замена символов "+" пробелами;
- выделение отдельных пар *имя=знанение>* разделенных ограничителем &;
- выделение из каждой пары *имя=значение* имени и значения соответствующего поля формы.

Программа декодирования HTML-формы может выглядеть, например, так:

```
#!/usr/bin/perl
# Декодирование данных формы, переданных методом GET $form_data = $ENV{'QUERY_STRING'};
# преобразование цепочек %hh в соответствующие символы $form_data-=- s/%(..)/pack ("C", hex
($1))/eg; i преобразование плюсов в пробелы $form_data =~ tr/+/ /;
# разбиение на пары имя=значение @pairs = split (/&/, $form_data);
# выделение из каждой пары имени и значения поля формы и сохранение
# их в ассоциативном массиве $fom_fields
foreach $pair (@pairs)
{
($name, $value)=split(/=/,$pair);
$ form_fields{$name}=$value; }
```

Если данные формы переданы методом POST, то в приведенном тексте следует заменить оператор присваивания

```
$form_data = $ENV{'QUERY_STRING'};
```

оператором

```
read(STDrN,$fom_data,$ENV{'CONTENT_LENGTH' }} ;
```

считывающим из стандартного ввода программы CONTENT_LENGTH байтов, составляющих содержимое запроса клиента, в переменную $form_data.

В приведенном примере используются две новые функции: packQ и hex(). Поясним их назначение прежде, чем перейти к обсуждению текста программы.

Функция

```
pack template, list
```

упаковывает список значений list в двоичную структуру по заданному шаблон у template.

Аргумент template представляет собой последовательность символов, определяющих формат представления пакуемых данных:

- а/А Текстовая строка, заполненная нулями/пробелами
- ь/в Двоичная строка, значения расположены в порядке возрастания/ убывания
- с/с Обычное символьное значение/ Символьное значение без знака
- f/d Значение в формате с плавающей точкой одинарной/двойной точности
- ь/н Шестнадцатеричная строка, младший/старший полубайт первый
- i/i Целое со знаком/ без знака
- I/L Значение типа long со знаком/без знака
- П/N Значение типа short/long с "сетевым" порядком байтов ("старший в старшем")
- P/U Указатель на строку/Ш-кодированная строка s/s Значение типа short с$> знаком/без знака
- v/v Значение типа short/long с VAX-порядком байтов ("старший в младшем")
- х/х Нулевой байт/резервная копия байта
- @ Заполнение нулевыми байтами (до абсолютной позиции)

За каждым символом может следовать число, обозначающее счетчик применений данного символа в качестве формата. Символ * в качестве счетчика означает применение данного формата для оставшейся части списка.

```
$x = pack "cccc", 80, .101, 114, 108; $x = pack "c4", 80, 101, 114, 108;
$x = pack "B32", "01010000011001010111001001101100";
```

```
$x = pack "H8", "5065726C";
$x = pack "H*", "5065726C"; ' ·
$x = pack "cB8H2c",80,"01100101",12,108;
```

Значение переменной $x во всех случаях равно "perl". Функция

```
hex expr
```

Интерпретирует аргумент expr как шестнадцатеричную строку и возвращает ее десятичное значение.

В тексте программы примера 15.4 все представляется очевидным. Разберем только наиболее насыщенную строку

```
$fonn_data =~ s/%(..)/pack ("C", hex ($l))/eg;
```

Образец для поиска задан в виде регулярного выражения %(..). Этому образцу удовлетворяет произвольная последовательность вида %xy, где x, y — любые символы. В результате кодирования данных в качестве x, y могут появиться только шестнадцатеричные цифры, поэтому можно не задавать более точный, но менее компактный шаблон %([0-9A-Fa-f][0-9A-Fa-f]j. Часть выражения заключена в скобки (..). При нахождении подходящего фрагмента %hh его часть, содержащая шестнадцатеричное число hh, сохраняется *в* переменной, которая затем будет использована в качестве аргумента функции hex($ij для преобразования в десятичное значение. Функция pack упакует это десятичное значение *в* двоичную структуру, которая в соответствии с шаблоном "c" будет интерпретироваться как символ. Этот символ заменяет в тексте найденную цепочку %hh.

После выделения и декодирования данных можно приступить к их обработке. Попробуем написать CGI-сценарий, обрабатывающий данные формы из примера 15.1.

Пример создания собственного CGI-сценария

Программа должна декодировать полученные данные, проверять заполнение обязательных полей формы и правильность подтверждения пароля, в зависимости от результатов проверки формировать документ для отсылки клиенту. Сохраним сценарий в файле /cgi-bin/registrar, cgi. Полный маршрут к данному файлу определяется параметрами конфигурации Web-сервера. Местоположение каталога cgi-bin обычно указывается относительно корня дерева документов Web-сервера, а не корневого каталога файловой системы. Например, если корнем является каталог /home/httpd/htmi/, то файл сцена-рия будет иметь маршрутное ИМЯ/home/httpd/html/cgi-bin/registrar.cgi, которое в запросе клиента будет указано как /cgi-bin /registrar.cgi. В первом приближении текст сценария может выглядеть следующим образом.

```
#!/usr/bin/perl
print "Content-type:text/html\n\n"; $method = $ENV{'REQUEST_METHOD'}; if ($method eq "GET") {
^
$form_data = $ENV{'QUERY_STRING'}; } else {
```

```
read (STDIN, $form_data, $ENV{'CONTENT_LENGTH'}); }
$form_data =~ s/%(..)/pack ("C", hex ($l))/eg; $form_data =~ tr/+/ /; @pairs = split (/&/,
$form_data); foreach $pair (Spairs) {
($name, $value)=split(/=/,$pair);
$FORM{$name}=$value; /
1 ("
Шроверка заполнения обязательных полей
if (!$FORM{'regname'} I I !$FORM{'passwordl'}) { print «goback
<html>
<headxtitle>HencxiiHbie n.aHHbie</title></head>
<body><h2>M3BHHHTe, Вы пропустили обязательные данные</п2>
<br>
<a href="http://www.klf.ru/welcome.Мл11">Попробуйте еше раз, пожалуйста</a>
</body>
</html> goback ;}
#Проверка правильности ввода пароля elsif ($FORM{'passwordl1} eq $FORM{'password2'}){
print«confirmation <htral>
<headxtitle>no3flpaBnHeM!</titlex/head> <bойу><b2>Поздравляем! </h2xbr>
Ваша регистрация прошла успешно. Вы можете пользоваться нашей библиотекой. Спасибо за
внимание.
</body>
</html>
confirmation
;}
else {
print«new_form
<htmlxhead><title>Oira6Ka при вводе парс«w</titlex/head>
<bойуХп3>Введенные Вами значения пароля не совпадают
<brxform method="get" action="/cgi-bin/registrar.cgi">
<pre>
Введите пароль: <input type="password" name="passwordl">
Подтвердите пароль: <input type="password" name="password2">
</pre>
new_form
foreach $key ( keys %FORM) {
if ($key ne "passwordl" && $key ne "password2") {
print "<input type=\"hidden\" name=$key value=$FORM{$key}>\n";
} } print «EndOfHTML
```

```
<br><br>
<input type="submit" value="OK"> <input type="reset" value="Отменить">
<i/form></body></html> EndOfHTML ;)
```

После вывода строки заголовка осуществляется считывание переданной серверу информации в переменную $form_data. В зависимости от метода передачи, эта информация считывается из переменной среды QUERY_STRING (метод GET) или из стандартного ввода программы (метод POST).

Считанная информация декодируется и помещается в ассоциативный массив %FORM.

Отсутствие обязательных данных – регистрационного имени и пароля – проверяется с помощью условия if (! $FORM{' regname'} I I ! $FORM{ 'passwordl'}).

В случае отсутствия необходимых данных формируется виртуальный HTML-документ, предлагающий повторить попытку, который и посылается клиенту (рис. 15.4).

Рис 15.4. Ответ сервера в случае отсутствия обязательной информации

При выводе этого документа в операции print использована конструкция "документ здесь". Она позволяет использовать внутри себя символы, которые при заключении в обычные двойные кавычки необходимо маскировать символом "\", например, сами, двойные кавычки ", символы "@", ••$", "%".

Условие elsif ($FORM{ 'passwordl[1] } eq $FORM{ 'password2 ' }} предназначено для проверки совпадения дву?с копий введенного пользователем пароля. Если значения совпадают, то пользователю посылается сообщение, подтверждающее успешную регистрацию (рис. 15.5).

Рис 15.5. Подтверждение регистрации

В противном случае формируется HTML-документ, предлагающий ввести пароль повторно (рис. 15.6). Этот новый документ содержит форму, в состав которой входят два видимых поля типа "password" –

для ввода и подтверждения пароля, и скрытые поля типа "hidden" – для сохранения остальных данных, введенных при заполнении исходной формы. Каждое скрытое поле новой формы наследует у соответствующего поля исходной формы атрибуты name и value. Если эти данные не сохранить, то их придется вводить заново, принуждая пользователя повторно выполнять уже сделанную работу. Информация, сохраненная в скрытых полях, невидима пользователю и недоступна для изменения.

Рис 15.6. Повторное приглашение для ввода пароля

Культура Perl допускает различные уровни владения языком. В рассмотренном варианте использован минимальный набор средств. Очевидно, что часть кода, например, декодирование, требуется при обработке не только данной, но и любой другой формы. Естественным шагом в развитии исходного варианта сценария является выделение этой части в отдельную подпрограмму и предоставление доступа к ней другим сценариям. Для этого преобразуем исходный текст в соответствии с планом, изложенным в примере 15.7.

1. Часть исходного кода может быть использована другими CGI-программами. Преобразуем ее в отдельный модуль, сохраняемый в файле CGI__UTILS.pm.

```
package CGI_UTILS; require Exporter;
@ISA = qw(Exporter);
^EXPORT = qw(print_header process_input);
# Подпрограмма вывода заголовка ответа sub print_header {
print "Content-type: text/html\n\n"; } .
# Подпрограмма декодирования данных форьы sub process_input {
my ($form_ref)=(?_.;
my ($ form_data,@pai rs);
my ($temp)=""; /
if ($ENV{'REQUEST_METHOD'} eq 'POST') {
read(STDIN,$form_data,$ENV{'CONTENT_LENGTH'}); } else {
$form_data=$ENV{'QUERY_STRING'}; }
$form_data=~s/%(..)/pack("c",hex($1))/ge; $form_data=~tr/+/ /; $form_data=~s/\n/\0/g;
@pairs=split(/&/,$form_data); foreach $item(@pairs) {
($name,$value)=spl£t (/=/,$item) ; if (!defined($forih_ref->{$name})) <
```

```
$ form_ref->{Sname}=$value; } else {
$form_ref->{$name} .= "\0$value"; } }
foreach $item (sort keys %$form_ref) { $temp.=$item."=".$form_ref->{$item}."&";}
return($temp); } 1;
```

2. Текст основного сценария обработки формы registrar.cgi преобразуем следующим образом:

```
#!/usr/bin/perl use cgi_utils;
my 8FORM, $file_rec; $file_rec=&process_input(\%FORM); ^Проверка заполнения обязательных полей
#if {$FORM{'regname'} eq "" || $FORM{'passwordl'} eq "") { if (!$FORM{'regname'} ||
!$FORMTpasswordl'}) {
print "Location: /goback.html\n\n"; }
#Проверка правильности ввода пароля
elsif ($FORM{'passwordl'} eq $FORM{'password2'}}{
print "Location: /confirmation.html\n\n";
open (OUTF, "»users");
print OUTF $file_rec, H\n";
close OUTF }
else {
&print_header; print«new_form
<html>
<headxtitle>OtiM6Ka при вводе паролtt</titlex/head>
<boйухп3>Введенные Вами значения пароля не совпадают
<br><forro method="get" action="/cgi-bin/registrar.cgi">
<pre>
Введите пароль: <input type="password" name="passwordl">
Подтвердите пароль: <input type="password" name="password2">
</pre> new_form / foreach $key ( keys %FORM) {
if ($key ne "passwordl" && $key ne "password2") {
print "<input type=\"hidden\'J name=$key value=$FORM{$key}>\n";
} } print«EndOfHTML
<brxbr>
<input type="submit" value="OK">
<input type="reset" value="OTMeHHTb">
</form>
</body>
</html> EndOfHTML
} exit
```

3. В исходном варианте сценария в качестве ответов сервера при получении неполных данных и для подтверждения регистрации пользователя формируются виртуальные HTML-документы. В этом нет необходимости, так как они содержат только статическую информацию. Соответствующие фрагменты сценария преобразуем в HTML-код ротовых документов, которые сохраним в отдельных файлах. В основном сценарии в качестве ответа сервера возвращаются ссылки на эти документы.

Файл confirmation.html содержит документ, посылаемый клиенту в качестве сообщения об успешной регистрации:

```
<html>
<head><title>Поздравляем!</title></head> <body><h2>Поздравляем! </h2xbr>
Ваша регистрация прошла успешно. Вы можете пользоваться нашей библиотекой. ..- -
<br> /
Спасибо за внимание.
</body>
</html>
Файл goback.html содержит документ, посылаемый клиенту при получении неполных данных:
<html>
<head><title>Неполные данные</^^1ex/пеай>
<body><h2>HsBHHMTe, Вы пропустили обязательные данные</b2>
<br>
<a href="">Попробуйте еще раз, http://www.klf.ru/welcome.Ылп1
4>пожалуйста</a>
</body>
</htral>
```

В приведенном тексте появились некоторые новые элементы, которые необходимо пояснить.

Подпрограмма process_input модуля cgi_utiis.pm передает декодированные данные через вызываемый по ссылке параметр – ассоциативный массив. [Кроме того, она возвращает при помощи функции return () те же данные, но в виде строки, состоящей из пар имя=значение, разделенных символом "&". Обратите внимание на то, как подпрограмма вызывается в основной программе:

```
$file_rec=sprocess_input(\%FORM);
```

В качестве аргумента ей передается ссылка на ассоциативный массив. В тексте подпрограммы появилась проверка наличия полей формы с совпадающими именами и разными значениями:

```
if (!defined($form_ref->{$name})) {
$form_ref->{$name}=$value; }
else { }}
```

Этот фрагмент необходим для того, чтобы правильно обработать следующую ситуацию из нашего примера. Выбраны несколько переключателей, определяющих языки, которыми владеет пользователь: русский, английский, французский. Так как соответствующие элементы формы имеют одинаковые имена name=language, ТО без проверки В ассоциативный массив%fcrm_ref, куда помещаются обработанные данные, попадет только информация от последнего обработанного Элемента name=language value=french. В Подобном случае обычное присваивание заменяется операцией присваивания с конкатенацией

```
$form_ref->{$name} .= "\0$value",
```

которая к переменной $fom_ref->{$name) добавляет нулевой символ и значение $value.

В основной программе registrar.cgi обратим внимание на то, как передается ссылка на готовый HTML-документ. Для этого вместо заголовка content-type: text/html выводится заголовок Location: URL, сообщающий серверу адрес документа.

Еще один новый элемент в основной программе — сохранение данных в файле с именем users.

Модуль CGI.pm

Пример, рассмотренный выше, демонстрирует наивный подход, когда кажется, что все необходимые программы надо писать самостоятельно с самого начала. Но программирование CGI — это такая область, в которой Per] давно и активно применяется, и многое из того, что может потребоваться, уже давно кем-то написано. Надо только найти и использовать. В. данном разделе мы сделаем краткий обзор одного из таких готовых средств, предназначенных для поддержки разработки CGI-приложений.

Модуль CGI.pm, созданный Линкольном Штейном, входит в состав дистрибутивного комплекта Perl, начиная с версии 5.004, и его даже не нужно специально инсталлировать.

Этот модуль содержит большой набор функций для создания и обработки HTML-форм. Мы посвятили значительную часть предыдущего раздела изучению многочисленных тэгов, чтобы затем написать HTML-код для создания формы в примере 15.1. Модуль CGI позволяет сделать то же самое, но без использования HTML. С его помощью можно описать форму на языке Perl, используя вместо тэгов обращения к функциям модуля. В результате получится не документ HTML, а сценарий на языке Perl, который при вызове будет "на лету" генерировать HTML-форму и передавать серверу для отправки клиенту.

Модуль CGI является не просто модулем, а классом, что позволяет использовать преимущества объектно-ориентированного подхода. Модуль предоставляет пользователю на выбор два вида интерфейса с самим собой: процедурно-ориентированный и объектно-ориентированный.

При использовании процедурно-ориентированного способа работы с модулем CGI функции модуля нужно явным образом импортировать в пространство имен вызывающей программы, а затем обращаться к ним

как обычно. В этом случае в вызывающей программе должны быть строки, аналогичные следующим:

```
#!/usr/bin/perl ; use CGI qw/:standard/; \ print header(), \
start_html('Пример формы'),
hi('Пример формы'),
```

Директива use импортирует в пространство имен вызывающей программы некоторый стандартный набор функций. Помимо него, существуют другие наборы функций модуля CGI. Их можно импортировать, указав имя соответствующего набора в списке импорта директивы use. Имена всех наборов можно просмотреть в файле CGI.pm, где они содержатся в хеш-массиве

```
%EXPORT_TAGS,
```

Функции header (), start_html О , hi () ЯВЛЯЮТСЯ функциями модуля CGI. Они будут рассмотрены ниже.

При использовании объектно-ориентированного интерфейса в директиве use вызывающей программы не нужно указывать список импортируемых имен функций. В этом случае взаимодействие с модулем CGI осуществляется через объект класса CGI, который нужно создать в вызывающей программе при помощи конструктора new (). Объектно-ориентированный вариант приведенного выше фрагмента выглядит следующим образом:

```
#!/usr/bin/perl
use CGI;
Squery = new CGI;
print $query->header(),
$query->start_html {'Пример формы'),
$query->hl ('Пример формы) , 1
```

Замечание

Функции модуля CGI.pm являются методами класса CGI. Для того чтобы их можно было вызывать и как функции, и как методы, синтаксис не требует в качестве обязательного первого параметра указывать объект класса CGI. Поэтому в качестве функций к ним можно обращаться обычным образом, а как к объектам — только используя форму $object->method ().

Модуль как мы отметили выше, содержит большой набор методов, и в наши планы не входит их подробное изучение. Документация, входящая в состав самого модуля, достаточно подробно описывает его компоненты. Чтобы получить представление о работе модуля CGI, создадим с его помощью небольшой сценарий. Для этого вернемся к рассмотрению формы из примера 15.1.

Будем для определенности использовать традиционный процедурно-ориентированный интерфейс. Рассмотрим следующий сценарий.

```
#!/usr/bin/perl
use CGI qw(:standard);
print header;
print start_html('Пример формы'),
h2('Регистрационная страница Клуба любителей фантастики'),
'Заполнив анкету, вы сможете пользоваться нашей электронной
"^библиотекой.',
br,
start_form,
"Введите регистрационное имя:",textfield('regname'),
P.
"Введите пароль: ", password_field(-name=>'passwordl',
-maxlength=>'8'),
P,
"Подтвердите пароль: ", password_field(-name=>'password2',
-maxlength=>'8'),
P/-
"Ваш возраст",
P'
radio_group(-name=>'age',
-value=>['lt20','20_30', '30_50', 'gt50'],
-default=>'Lt20',
-labels=>{'It20'=>'flo 20',20_30'=>'20-30', 4> '30_50'=>'30-50','gt50'=>'старше 50'}), I
br,br,
"На каких языках читаете:",
checkbox_group(-name=>'language', 4> -values=>
4>
['русский', 'английский,'французский', 'немецкий'], 1
^ -defaults=>['русский']), br,br,
"Какой формат данных является для Вас предпочтительным ", br, popup_menu(-name=>'type',
-values=>['Plain text','PostScript','РОГ']), br,br, \
N
"Ваши любимые авторы:", -~-_. x
br„
textarea(-name=>'wish', -cols=>40, -rpws=>3),
br,
subrai t{-name=>'OK'), reset{-name=>'Отменить'),
end_form,
```

```
hr;
if (param0) { print
"Ваше имя: ",em(param('regname')),
P,
"Ваш возраст: ", em(param('age')),
P, '
J
"Вычитаете на языках: ",em(join(", ",param('language'))),
P,
"Предпочтительный формат данных для Вас: ",em(param ('type')),
P,
"Ваши любимые авторы: ", em(join(", ",param('wish'))), 1
hr; } print end_html;
```

Обсудим приведенный текст. Директива use, как отметили выше, осуществляет импорт стандартного набора функций модуля CGI.pm в пространство имен вызывающего пакета. В самом сценарии на месте тэгов исходного HTML-кода стоят обращения к функциям модуля: каждому тэгу соответствует вызов функции. Вызов функции модуля CGI можно осуществлять двумя способами: с использованием позиционных параметров

```
print textfield('regname','начальное значение',50,80); с использованием именованных параметров
мы
print textfield(-name=>'regname',
-default=>'начальное значение",
-size=>50,
-maxlength=>80);
```

Обработка позиционного параметра внутри функции зависит от его места в списке параметров. Обработка именованного параметра не зависит от его места в списке параметров. Функции модуля CGI могут иметь большое число параметров, порядок следования которых трудно запомнить, поэтому в этом модуле была реализована возможность вызова функций с именованными параметрами. Кроме того, применение именованных параметров делает текст программы более понятным. В тексте примера функции вызываются с именованными параметрами, если параметров больше одного. Познакомимся с функциями, использованными в примере.

Функция header о без параметров создает для виртуального ответа сервера стандартный HTTP-заголовок Content-Type: text/html и вставляет после него необходимую пустую строку. Параметры позволяют задать дополнительную информацию для заголовка, например, указать другой медиа-тип содержимого или код ответа, посылаемый браузеру:

```
print header(-type=>'image/gif',
-status=>'404 Not Found 1);
```

Функция start_htmi () создает HTML-заголовок и начальную часть документа, включая открывающий тэг <booy>. При помощи параметров функции можно задать дополнительные тэги внутри тэга <HEAD>, а также значения атрибутов. Все параметры являются необязательными. В примере функция start_htmio вызвана с одним позиционным параметром, определяющим название документа.

Модуль CGI содержит методы (функции) для поддержки многих тэгов HTML2, HTML3, HTML4 и расширений, используемых в браузерах Netscape. Тэгам соответствуют одноименные методы модуля CGI.pm, имена которых записываются при помощи символов нижнего регистра. Если при этом возникают конфликты имен, в названия методов следует вводить символы верхнего регистра, как, например, в следующих случаях.

- Название тэга <TR> совпадает с именем встроенной функции tr(). Имя соответствующего метода записывать в виде TR () или тг ().
- Название тэга <PARAM> совпадает с именем собственного метода модуля CGI paramo. Для обозначения метода, соответствующего тэгу, использовать ИМЯ PAKAM ().
- Название тэга <SELECT> совпадает с именем встроенной функции select (). Для обозначения метода использовать имя Select ().
- Название тэга <зив> совпадает с именем ключевого слова объявления функции sub. Для обозначения метода использовать имя sub ().

Тэгам, имеющим атрибуты, соответствуют методы, имеющие в качестве первого аргумента ссылку на анонимный хеш-массив. Ключами этого хеш-массива являются имена атрибутов тэга, а значениями — значения атрибутов.

Методы, соответствующие тэгам, и методы, предназначенные длячгенерирова-ния других элементов HTML-документа, возвращают строки, содержащие соответствующие элементы. Чтобы эти строки попали в создаваемый документ, их нужно вывести, как это делается в примере при помощи функции print.

В примере использованы следующие методы, соответствующие тэгам HTML.

- Функция H2 соответствует тэгу <H2>. Она определяет, что ее аргумент является в документе заголовком второго уровня.
- Функция Ьг соответствует тэгу
 и обозначает, что последующий текст размещается с начала новой строки.
- Функция р соответствует тэгу <р> и обозначает начало абзаца.
- Функция hr соответствует тэгу <т> и обозначает горизонтальную линию, разделяющую документ на части.

- Функция em соответствует тэгу <ет> и обозначает, что ее аргумент в документе должен быть выделен курсивом.

Следующие функции используются для создания формы и ее элементов.

- Функция start_form соответствует тэгу <FORM>. Она может иметь три параметра start_form

```
(-method=>$method,
-action=>$action,
-encoding=>$encoding);
```

при помощи которых можно задать метод передачи формы Web-серверу (-method), программу, предназначенную для обработки формы (-action), и способ кодирования данных (-encoding). Все параметры являются необязательными. По умолчанию используются значения

```
method: POST;
action: данный сценарий;
encoding: application/x-www-form-urlencoded.
```

- Функция end_form создает закрывающий тэг </FORM>.
- Функция textfieid соответствует тэгу <INPUTE TYPE=TEXT>. Она имеет следующий синтаксис

```
textfieid (-name=>' field__name',
-default=>'starting value',
-size=>50,
-maxlength=>80);
```

Параметры соответствую^ атрибутам тэга. Обязательным является первый параметр.

```
/ x. __ _ _^/
```

- Функция password_fieid соответствует тэгу <INPUTE TYPE=PASSWORD>. Ее синтаксис:

```
password_field(-name=>'secret',
-value=>'starting value',
-size=>8,
-maxlength=>12);
```

Параметры имеют тот же смысл, что и одноименные атрибуты соответствующего тэга. Обязательным является первый параметр.

- Функция radio_group служит для создания группы "радиокнопок" — элементов, задаваемых тэгом <INPUTE TYPE=RADIO>. Ее синтаксис имеет следующую форму

```
radio_group(-name=>'group_name',
-values=>['bim',Jbam','bom'],
-default=>'bom',
-linebreak=>'true',
-lab.els=>\%labels);
```

Первый аргумент является обязательным, соответствует одноименному атрибуту тэга. Второй аргумент тоже обязательный и задает значения элементов. Эти значения отображаются в качестве названий кнопок. Он должен быть ссылкой на массив. Остальные аргументы являются необязательными. Третий аргумент задает кнопку, которая выбрана по умолчанию. Если значение четвертого аргумента 'true, каждая следующая кнопка группы размещается в начале новой строки. Пятым аргументом является ссылка на хеш-массив, который связывает значения, присвоенные кнопкам, с метками, которые отображаются в виде названий кнопок. Если аргумент не задан, то в качестве названий отображаются сами значения.

- Функция checkbox_group служит для создания группы элементов-переключателей, задаваемых тэгом <INPUTE TYPE= CHECKBOX>.

```
checkbox_group(-name=>'group_name',
-values=>['bim','bam','bom'],
-default=>['bim','bom'],
-linebreak=>'true',
-labels=>\%labels);
```

Аргументы имеют тот же смысл, что и одноименные аргументы функции radio_group. Поскольку в группе переключателей можно одновременно выбрать несколько элементов, третий аргумент может быть или одиночным элементом, или ссылкой на массив, содержащий список значений, выбранных по умолчанию. Обязательными являются первый и второй аргументы.

- Функция popup_menu служит для создания меню, задаваемого при помощи тэга <SELECT>. Имеет следующий синтаксис:

```
popup_menu(-name=>'menu_name',
-values=>['bim', 'bam','bom'],
-default=>'bom, ᶜ
-labels=>\%labels);
```

Первый аргумент задает имя меню. Второй аргумент является ссылкой на массив, содержащий список значений, присвоенных элементам меню. Первый и второй аргументы обязательны, остальные – нет. Третий аргумент задает элемент меню, выбранный по умолчанию. Четвертый аргумент является ссылкой на хеш-массив. Хеш-массив значению каждого элемента меню ставит в соответствие строку,

которая будет отображаться в меню для этого элемента. Если четвертый аргумент отсутствует, то для каждого элемента меню отображается его значение, заданное вторым аргументом.

- Функция textarea соответствует тэгу <TEXTAREA>, задающему в документе текстовое поле для ввода многострочного текста. Имеет следующий синтаксис

```
textarea(-name=>'region',
-default=>'starting value',
-rows=>10,
-columns=>50);
```

Первый параметр, задающий имя элемента формы <TEXTДКЕА>, является обязательным, остальные – нет. Второй параметр задает строку, отображаемую по умолчанию. Третий и четвертый параметры задают соответственно число строк и столбцов, отображаемых в текстовом поле.

- Функция submit соответствует тэгу <INPUT TYPE=SUBMIT>, задающему кнопку передачи. Ее синтаксис:

```
print $query->submit(-name=>'button_name', -value=>'value');
```

Первый параметр является необязательным. Он задает имя кнопки, которое отображается в качестве ее названия. Нужен только для переопределения названия и в тех случаях, когда надо различать несколько имеющихся кнопок передачи. Второй параметр тоже необязательный. Он задает значение, которое посылается в строке запроса при щелчке на этой кнопке.
Submit

- Функция reset соответствует тэгу <INPUT TYPE=RESET>, задающему кнопку сброса. Может иметь параметр, переопределяющий название **Reset,** отображаемое по умолчанию.

- Функция end_htmi завершает HTML-документ, добавляя тэги </BODY> </HTML>.

Пример 15.8 содержит также код, который не связан с созданием формы Он состоит из одного условного оператора, в котором в качестве условия используется значение, возвращаемое функцией paramo. Эта функция используется также внутри блока условного оператора. Разберем для чего она применяется. При помощи функции paramo модуля CGI можно выполнить следующие действия.

- Получение списка имен параметров, переданных сценарию. Если сценарию переданы параметры в виде списка пар "имя=значение" функция paramo без аргументов возвращает список имен параметров сценария:

```
@names = param;
```

- Получение значений именованных параметров. Функция paramo с единственным аргументом – именем параметра, возвращает значение этого параметра. Если параметру соответствует несколько

значений, функция param о возвращает список этих значений:'

```
@values = param('language'); в противном случае – одно значение:
$value = param('regname');
```

- Задание значений параметров.

```
param(-name => 'language', -values => ['russian', 'english', 'french']);
```

Можно задавать значения параметров, используя вызов функции param о в форме с позиционными параметрами, но тогда нужно знать порядок следования этих параметров:

```
param ('language', 'russian', 'english', 'french');
```

При помощи функции param о можно устанавливать начальные значения элементов формы или изменять ранее установленные значения.

Часть сценария, предшествующая условному оператору, предназначена для создания формы из примера 15.1. Заключительная часть, состоящая из условного оператора, обрабатывает заполненную и отправленную Web-серверу форму. Это происходит потому, что по умолчанию приложением, обрабатывающим форму, является данный сценарий (см. описание start__form). Таким образом, в одном сценарии содержится код, и создающий форму, и ее обрабатывающий.

Сохраним код, приведенный в примере 15.8, в файле welcome.cgi. Этот файл можно поместить на Web-сервере в стандартный каталог cgi-bin, предназначенный для хранения CGI-сценариев. Предположим, что Web-сервер имеет Internet-адрес Если из удаленного браузера послать запрос по адресу то Web-сервер, получив запрос, выполнит сценарий welcome, cgi. Сценарий "на лету" создаст HTML-документ, содержащий форму, и передаст его Web-серверу, который отправит документ браузеру. Браузер, получив документ, отобразит его. **www.klf.ru, http://www.klf.ru/cgi-bin/welcome.cgi.**

После заполнения формы и нажатия кнопки данные формы будут вновь отправлены Web-серверу, который передаст их для обработки все тому же сценарию welcome.cgi. Сценарий "на лету" создаст новый HTML-документ с учетом полученных данных и через сервер направит его браузеру. Браузер отобразит новый документ.

Сценарий welcome.cgi можно передать для выполнения интерпретатору perl, а результат вывести в файл, чтобы посмотреть, как вызовы функций модуля CGI преобразуются в тэги HTML-документа. Документ HTML, созданный сценарием welcome.cgi, имеет следующий вид.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN"> <HTMLXHEADXTITLE>npMMep <J>opMbi</TITLE>
</HEADXBODY>
<H2>Регистрационная страница Клуба любителей фантастики</H2>
Заполнив анкету, вы сможете пользоваться нашей электронной библиотекой,
```

```
<BR>
<FORM METHOD="POST" ENCTYPE="application/x-www-form-urlencoded">
Введите регистрационное имя:<1ЫРит TYPE="text" NAME="regname" ><P>
Введите пароль: <INPUT TYPE="password" NAME="passwordl" MAXLENGTH=8><P>
Подтвердите пароль: <INPUT TYPE="password" NAME="password2" MAXLENGTH=8>
<P>Ваш возраст<P>
<INPUT TYPE="radio" NAME="age" VALUE="lt20" CHECKEO>до 20
<INPUT TYPE="radio" NAME="age" VALUE="20_30">20-30
<INPUT TYPE="radio" NAME="age" VALUE="30_50">30-50
<INPUT TYPE="radio" NAME="age" VALUE="gt50">CTapnie 50
<BRXBR>На каких язьпсах читаете:
<INPUT TYPE="checkbox" NAME="language" УАШЕ="русский" CHECKEП>русский
<INPUT TYPE="checkbox" NAME="language" УАШЕ="английский">английский
<INPUT TYPE="checkbox" NAME="language" УАьиЕ="французский">фанцузский p
<INPUT TYPE="checkbox" NAME="language" VALDE="немецкий"Жемецкий
<BRXBR>
Какой формат данных является для Вас предпочтительным
<BRXSELECT NAME="type">
<OPTION VALUE="Plain text">Plain text
<OPTION VALUE="PostScript">PostScript
<OPTION VALUE="PDF">PDF
</SELECT>
<BRXBR>
Ваши любимые авторы:
<BRXTEXTAREA NAME="wish" ROWS=3 COLS=40X/TEXTAREA>
<BR>
<INPUT TYPE="sxobmit" NAME="OK" VALUE="OK">
<INPUT TYPE="reset" VALUE="Отменить">
<INPUT TYPE="hidden" NAME=".cgifields" VALUE="language">
<INPUT TYPE="hidden" NAME=".cgifields" VALUE="age">
</FORM>
<HRX/BODYX/HTML>
```

В действительности документ, созданный сценарием welcome. cgi, состоит из небольшого количества длинных строк, что связано с тем, как они формируются методами модуля CGI. Поэтому реально сформированный текст для удобства представлен в более структурированном виде. Но это единственное изменение, не влияющее на смысл автоматически созданного документа.

Вопросы для самоконтроля

1. Что обозначает аббревиатура CGI?
2. Что такое Web-сервер?
3. Что такое клиент Web-сервера?
4. Для чего нужна HTML-форма?
5. Какую первую строку должна выводить CGI-программа?
6. Какие методы передачи данных формы Web-серверу вы знаете? В чем их особенности?
7. Перечислите элементы управления, которые может содержать форма.
8. Как кодируется информация, передаваемая CGI-программе?
9. В чем заключается специфика CGI-сценариев?
10. Каким образом CGI-программа имеет доступ к переменным среды CGI?
11. Какая информация содержится в переменной среды QUERY_STRING?
12. Для чего предназначен модуль CGI.pm?

Упражнения

1. Запишите текст примера 15.8 в обозначениях, использующих объектно-ориентированный интерфейс с модулем CGI.

2. Используя модуль CGI, дополните пример 15.8 кодом, осуществляющим проверку введенных данных, как это сделано в примере 15.76.

3. Ниже приведены тексты трех примеров, входящих в отдельный дистрибутив модуля CGI.pm. Разберите, что делают эти сценарии.

- a) clickable_image.cgi

```
#!/usr/bin/perl
use CGI;
$query = new CGI;
print $query->header;
print $query->start__html("A Clickable Image");
print «END;
<H1>A Clickable Image</Hl>
</A> /
END |
print "Sorry, this isn't very exciting!\n";
print $query->startform;
print $query->image_button('picture',"./wilogo.gif"); print "Give me a: ", $query->popup_rnenu
('letter',
['A', 'B', 'C', 'D', 'E', 'W]),"\n"; print "<P>Magnification: ", $query-
>radio_group('magnification',
```

```
['IX,'2X',MX','20X']),"\n"; print "<HR>\n"; if ($query->param) {
print "<P>Magnification, <EM>",$query->param('magnification'),
"</EM>\n";
print "<P>Selected Letter, <EM>",$query->param( 'letter'),"</EM>\n"; ($x,$y) = ($query-
>param('picture.x'),$query->param('picture.y')) ; print "<P>Selected Position <EM>($x,$y)
</EM>\n";
}
print $query->end_html;
```

- 6) quadraphobia.cgi

```
#!/usr/bin/perl use CGI qw/:standard/; print header,
start_html('QuadraPhobia'),
hi('QuadraPhobia'),
start_form(),
image_button(-name=>'square',
-src=>'red_square.gif,
-width=>200,
-height=>200,
-align=>MIDDLE),
end_form(); if (paramO) {
($x,$y) = (param('square.x'),param('square.y'));
$pos = 'top-left' if $x < 100 && $y < 100;
$pos = 'top-right' if $x >= 100 && $y < 100;
$pos = 'bottom-left' if $x < 100 && $y >= 100;
$pos = 'bottom-right' if $x >= 100 && $y >= 100;
print b("You clicked on the $pos part of the square."); }
print p,a({href=>'../source.html'},"Code examples"); print end_html ();
```

- в) popup.cgi

```
#!/usr/local/bin/perl
use CGI;
$query = new CGI;
print $query->header;
print $query->start_html('Popup Window');
if (!$query->param) {
print "<Hl>Ask your Question</Hl>\n";
print $query->startform(-target=>'_new');
print "What's your name? ",$query->textfield('name'); .
```

```
print "<P>What's the corobination?<P>",
$query->checkbox_group(-name=>'words',
-values=>['eenie','meenie','minie','moe'],
-defaults=>['eenie','moe']);
print "<P>What's your favorite color? ", $query->popup_menu(-name=>'color',
-values=>['red','green','blue','chartreuse']), "<P>";
print $query->submit; print $query->endform;
} else {
print "<Hl>And the Answer is...</Hl>\n";
print "Your name is <EM>",$query->param(name),"</EM>\n";
print "<P>The keywords are: <EM>",
join(", ",$query->param(words)),"</EM>\n"; print "<P>Your favorite color is <EM>",
$query->param{color),"</EM>\n";
}
print qq{<PXA HREF="cgi_docs.html">Go to the documentation</A>};
print $query->end_html;
```