CGI - модуль, реализующий функции Common Gateway Interface ОПИСАНИЕ ВЕРСИИ 2.56

НАИМЕНОВАНИЕ

CGI - модуль, реализующий функции Common Gateway Interface (CGI - общего интерфеска шлюзов)

ПОДДЕРЖИВАЕМЫЕ ПЛАТФОРМЫ

- Linux
- Solaris
- Windows

СИНТАКСИС и ПРИМЕР

```
# сценарий CGI, создающий заполняемую форму
# и выводящую полученные значения в окне браузера.
use CGI qw/:standard/;
print header,
      start_html('Простой пример'),
      h1('Простой пример'),
      start_form,
      "Ваше имя? ",textfield('name'),p,
      "Любимые слова?", p,
      checkbox_group(-name=>'words',
                  -values=>['Хрю','Гав','Мяу','Муу'],
                  -defaults=>['Хрю','Муу']), p,
      "Любимый цвет? ",
      popup_menu(-name=>'color',
               -values=>['красный','зеленый','синий','янтарный']),p,
      submit,
      end_form,
      hr;
 if (param()) {
     print "Ваше имя",em(param('name')),p,
          "Ваши любимые слова: ",em(join(", ",param('words'))),p,
          "Ваш любимый цвет",em(param('color')),
          hr;
 }
```

РЕФЕРАТ

Этот модуль perl использует объекты perl5 для облегчения создания форм и обработки их содержимого. Этот модуль определяет объекты CGI, логические сущности, которые содержат текущие параметры запроса и другие переменные состояния. Используя объектные методы CGI.pm, Вы можете анализировать ключевые слова и параметры, передаваемые Вашему сценарию, а также создавать интерактивные формы, начальные значения которых взяты из текущего запроса (сохраняя, соответственно, информацию о текущем состоянии). Данный модуль обеспечивает вызов функций, которые генерируют шаблоны HTML, что уменьшает вероятность ошибок при наборе текста и кодировании. Он также обеспечивает функциональность некоторых более продвинутых возможностей программирования интерфейса CGI, включая поддержку выгрузки файлов, поддержку cookie, стилевого оформления, страниц ,выталкиваемых сервером (push), и фреймов.

Модуль CGI.pm также обеспечивает простой функционально-ориентированный интерфейс для тех программистов, которым нет необходимости использовать объектно-ориентированные возможности.

Текущая версия CGI.pm доступна по адресам:

http://www.genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html
ftp://ftp-genome.wi.mit.edu/pub/software/WWW/

ОПИСАНИЕ

СТИЛЬ ПРОГРАММИРОВАНИЯ

Существует два типа стиля программирования с использованием CGI.pm,- объектно-ориентированный и функционально - ориентированный. При использовании объектно-ориентированного стиля Вы создаете один или более объектов CGI, а затем применяете к ним методы для создания различных элементов страницы HTML. Каждый объект CGI начинается со списка именованных параметров, который передается сервером Вашему сценарию CGI. Вы можете изменять объекты, сохранять их в файле или в базе данных, а затем вновь их создавать и использовать. В связи с тем, что каждый объект соответствует "состоянию" сценария CGI, и, следовательно, каждый список параметров объектов независим от другого, мы можем сохранять состояние сценария и позднее его восстанавливать.

Например, Вы можете создать простейшую HTML-страницу "Привет, мир!", используя объектно-ориентированный стиль:

```
#!/usr/bin/perl -w
use CGI;                          # Подключить модуль CGI
$q = new CGI;                     # создать новый объект CGI
print $q->header,                 # создать заголовок HTTP
      $q->start_html('Привет, мир'), # начало текста страницы
      $q->h1('Привет, мир'),      # Заголовок уровня 1
      $q->end_html;               # завершение документа HTML
```

При программировании с использованием функционально - ориентированного стиля имеется только один объект CGI, используемый по умолчанию, к которому Вы редко обращаетесь непосредственно. Вместо этого вызываете функции для получения параметров CGI, генерирования тэгов HTML, управления файлами cookie, и так далее. Это дает Вам возможность использования прозрачного программного интерфейса, но ограничивает Вас использованием только одного объекта CGI в рамках одного сценария. Пример, приведенный ниже, выводит ту же самую страницу, но использует функционально-ориентированный интерфейс. Главными отличиями от предыдущего примера являются: необходимость импортирования набора функций в наше пространство имен (обычно, "стандартные" функции), а также то, что не требуется создание объекта CGI.

```
#!/usr/local/bin/perl
use CGI qw/:standard/;          # загрузить стандартный набор функций CGI
print header,                 # создать заголовок HTTP
      start_html('Привет, мир'), # начало документа HTML
      h1('Привет, мир'),      # Заголовок уровня 1
      end_html;               # завершение документа HTML
```

Примеры в данном руководстве используют, главным образом, объектно-ориентированный стиль. Смотрите раздел ИСПОЛЬЗОВАНИЕ ФУНКЦИОНАЛЬНО -

ОРИЕНТИРОВАННОГО ИНТЕРФЕЙСА для получения дополнительных сведений об использовании функционально - ориентированного интерфейса CGI.pm

ВЫЗОВ ПОДПРОГРАММ CGI.PM

Большинству подпрограмм CGI.pm можно передать несколько аргументов, иногда до 20 необязательных аргументов! Для упрощения этого интерфейса, все подпрограммы используют стиль передачи именованного аргумента, который выглядит, к примеру, так:

```
print $q->header(-type=>'image/gif',-expires=>'+3d');
```

Каждое имя аргумента начинается с тире. Ни регистр, ни порядок следования аргументов в списке роли не играют. Т.е. -type, -Type и -TYPE означают одно и то же и воспринимаются одинаково. Фактически, только первый аргумент должен начинаться с тире. Если в первом аргументе использовано тире, CGI.pm присваивает тире всем последующим аргументам.

Вы не обязаны использовать тире вообще, если Вам этого принципиально не хочется. После создания объекта CGI, вызовите метод **use_named_parameters()** с ненулевым значением. Это скажет модулю CGI.pm что Вы намереваетесь использовать только именованные параметры:

```
$query = new CGI;
$query->use_named_parameters(1);
$field = $query->radio_group('name'=>'OS',
                             'values'=>['Unix','Windows','Macintosh'],
                             'default'=>'Unix');
```

Некоторые подпрограммы обычно вызываются только с одним аргументом. В этом случае Вы можете передать подпрограмме только один аргумент без явного указания его имени. Подпрограмма `header()` -она из них. В этом случае, единственный аргумент - тип документа.

```
print $q->header('text/html');
```

Другие подобные подпрограммы документированы далее.

Иногда именованные аргументы ожидают скалярное значение, иногда - ссылку на массив или хэш. Часто Вы можете передать подпрограмме аргумент любого типа, а подпрограмма выполнит наиболее подходящее действие. К примеру, подпрограмма `param()` используется для присвоения параметру CGI единичного или множественного значения. Два таких случая продемонстрированы ниже:

```
$q->param(-name=>'veggie',-value=>'помидор');
$q->param(-name=>'veggie',-value=>['помидор','п-пам-иддд-ор','картошка','ка-ка-ртош-ш-шка']);
```

Большое число процедур CGI.pm реально специально не определены в теле модуля, но генерируются автоматически по мере необходимости. Это т.н. HTML-сокращения, процедуры, которые генерируют тэги HTML для использования в

динамически создаваемых страницах. Тэги HTML имеют как атрибуты (пары атрибут="значение" внутри самого тэга), так и содержимое (часть между открывающими и закрывающими тэговыми операторными скобками). Для того, чтобы различать атрибуты и содержимое, CGI.pm использует соглашение о передаче атрибутов HTML как ссылки на хэш в качестве первого аргумента, и значений, если таковые существуют, в качестве последующих аргументов, в любом количестве, если таковые присутствуют. Это работает, к примеру, так:

```
Код                    Сгенерированный HTML
----                   --------------------
h1()                   <H1>
h1('какой-нибудь','текст');    <H1>какой-нибудь текст</H1>
h1({-align=>left});            <H1 ALIGN="LEFT">
h1({-align=>left},'заголовок');<H1 ALIGN="LEFT">заголовок</H1>
```

Тэги HTML описаны ниже более подробно.

Многие новички, начинающие пользоваться CGI.pm озадачены различием между вызовами HTML - сокращений, которые требуют наличия фигурных скобок, окружающих атрибуты тэга, и соглашением о вызове других процедур, которые руководят генерированием атрибутов и не используют фигурных скобок. Не смущайтесь. В качестве дополнительного удобства фигурные скобки являются необязательными во всех случаях, за исключением сокращений - генерирующих шаблонов HTML. Если Вам нравится, Вы можете использовать фигурные скобки во всех вызовах процедур, где требуется подстановка именованных аргументов. Например:

```
print $q->header( {-type=>'image/gif',-expires=>'+3d'} );
```

Если Вы используете при вызове Perl ключ **-w**, Вы будете предупреждены, что некоторые имена аргументов в CGI.pm конфликтуют со встроенными функциями Perl. Самым частым из таких предупреждений будет конфликт с аргументом - values, используемым для создания меню с множественным выбором, наборов радиокнопок и тому подобными конструкциями. Для того, чтобы избежать таких предупреждений, Вы можете воспользоваться одним из следующих советов:

1. **Используйте другое имя для аргумента, если возможно. К примеру для -values имеется алиасное имя -value.**
2. **Используйте заглавные буквы, например, так: -Values**
3. **Заключите имя аргумента в кавычки, например, так: '-values'**

Многие подпрограммы будут делать нечто полезное с именованным аргументом, который они не распознали в качестве ключевого. Например, Вы можете сформировать нестандартные поля HTTP-заголовка, передав их подпрограмме в качестве именованных аргументов:

```
print $q->header(-type  =>  'text/html',
                 -cost  =>  'Три денежки',
                 -annoyance_level => 'высокий',
                 -complaints_to   => 'мусорная корзина');
```

Эта конструкция даст в результате следующий нестандартный заголовок HTTP header:

```
HTTP/1.0 200 OK
Cost: Три денежки
Annoyance-level: высокий
Complaints-to: Мусорная корзина
Content-type: text/html
```

Заметьте, каким образом подчеркивания автоматически преобразуются в тире. Подпрограммы, генерирующие код HTML, выполняют различные виды преобразований.

Эта возможность позволяет адаптировать модуль к быстро изменяющимся стандартам HTTP и HTML.

СОЗДАНИЕ НОВОГО ОБЪЕКТА - ЗАПРОСА (ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ СТИЛЬ):

```
$query = new CGI;
```

Данное предложение проанализирует ввод (полученного при помощи любого из методов - как POST, так и GET) и сохранит переданные значения в объекте perl5 с именем $query.

СОЗДАНИЕ НОВОГО ОБЪЕКТА - ЗАПРОСА ИЗ ФАЙЛА

```
$query = new CGI(INPUTFILE);
```

Если Вы передаете методу `new()` дескриптор файла, он будет читать параметры из файла (или из STDIN, или из чего-нибудь еще). Файл может быть в любой из форм, описанных ниже (т.е. последовательность пар ТЭГ=ЗНАЧЕНИЕ, разделенных символами новой строки будет работать). Удобным является то, что именно такой тип файла создается при помощи метода `save()` (смотрите ниже). Могут быть записаны и восстановлены множественные записи в файл/из файла.

Борцам за чистоту языка Perl будет приятно узнать, что такой синтаксис воспринимает ссылки на дескрипторы файлов, или даже ссылки на шаблонные (glob) дескрипторы, которые являются "официальным" способом передачи дескриптора:

```
$query = new CGI(\*STDIN);
```

Вы также можете инициализировать объект CGI объектами FileHandle или IO::File.

Если Вы используете функционально - ориентированный интерфейс и хотите инициализировать состояние CGI из дескриптора файла, то это можно сделать при помощи функции **restore_parameters()**. Это позволит (повторно)

инициализировать объект CGI, установленный по умолчанию, указанным дескриптором файла.

```
open (IN,"test.in") || die;
restore_parameters(IN);
close IN;
```

Вы также можете инициализировать объект - запрос ссылкой на ассоциативный массив:

```
$query = new CGI( {'dinosaur'=>'Вася',
                   'song'=>'Рига-Москва',
                   'friends'=>[qw/Коля Петя Таня/]}
                 );
```

или правильно отформатированной, извлеченной из URL (очищенной) строкой запроса:

```
$query = new CGI('dinosaur=barney&color=purple');
```

или ранее инициализированным объектом CGI (в текущей версии модуля это приводит к клонированию списка параметров, но не других полей, определяющих объекты, таких как автоматическое преобразование текста в Esc - последовательности - autoEscape):

```
$old_query = new CGI;
$new_query = new CGI($old_query);
```

Для создания пустого запроса, инициализируйте его пустой строкой или пустым хэшем.:

```
$empty_query = new CGI("");
   -или-
$empty_query = new CGI({});
```

ИЗВЛЕЧЕНИЕ СПИСКА ПАРАМЕТРОВ ИЗ ЗАПРОСА:

```
@keywords = $query->keywords;
```

Если сценарий был вызван в качестве результата поиска <ISINDEX>, ключевые слова могут быть получены в виде массива при помощи метода `keywords()`.

ИЗВЛЕЧЕНИЕ КОЛИЧЕСТВА ПАРАМЕТРОВ, ПЕРЕДАННЫХ СЦЕНАРИЮ :

```
@names = $query->param;
```

Если сценарий был вызван со списком параметров (например, name1=value1&name2=value2&name3=value3), метод `param()` возвратит имена параметров в виде списка. Если же сценарий был вызван как <ISINDEX>, то будет возвращен единственный параметр, имеющий имя "keywords".

ПРИМЕЧАНИЕ: Начиная с версии 1.5, возвращаемый массив имен параметров будет иметь порядок, в котором они были переданы браузером. Обычно, порядок совпадает с порядком, в котором параметры определены в форме (однако, это четко не оговорено спецификацией, так что точное соответствие не гарантируется).

ИЗВЛЕЧЕНИЕ ЗНАЧЕНИЯ (ЗНАЧЕНИЙ) ОДНОГО ИМЕНОВАННОГО ПАРАМЕТРА:

```
@values = $query->param('foo');
        -или-
$value = $query->param('foo');
```

Передайте методу `param()` единичный аргумент, чтобы извлечь значение именованного параметра. Если значения параметра множественны (к примеру, выбраны несколько значений из прокручиваемого списка), Вы можете указать, что в качестве значения ожидается массив. В противном случае метод возвратит одиночное значение.

ПРИСВОЕНИЕ ЗНАЧЕНИЯ (ЗНАЧЕНИЙ) ИМЕНОВАННОМУ ПАРАМЕТРУ:

```
$query->param('foo','Это','массив,'полученных ','значений');
```

Такая конструкция присвоит именованному параметру 'foo' массив значений. Это один из способов изменить значение поля ПОСЛЕ вызова сценария. (Другим способом будет использование параметра -override, воспринимаемым всеми методами, генерирующими элементы формы).

`param()` также распознает стиль вызова именованного параметра, детально описанный далее:

```
$query->param(-name=>'foo',-values=>['это','массив','значений']);
                    -или-
$query->param(-name=>'foo',-value=>'значение');
```

ДОБАВЛЕНИЕ ДОПОЛНИТЕЛЬНЫХ ЗНАЧЕНИЙ К ИМЕНОВАННОМУ ПАРАМЕТРУ:

```
$query->append(-name=>'foo',-values=>['yet','more','values']);
```

Этот метод добавляет значение или список значений к именованному параметру. Значения добавляются в конец существующего параметра. Если параметр не существует, он автоматически создается. Обратите внимание на то, что этот метод распознает синтаксис вызова только именованного аргумента.

ИМПОРТИРОВАНИЕ ВСЕХ ПАРАМЕТРОВ В ПРОСТРАНСТВО ИМЕН:

```
$query->import_names('R');
```

Этот метод создает набор переменных в пространстве имен 'R', например, $R::foo, @R:foo. Для списков ключевых слов появится переменная @R::keywords. Если пространство имен не указано, метод по умолчанию назначает ему имя 'Q'. ПРЕДУПРЕЖДЕНИЕ: никогда ничего не импортируйте в 'main'; Вы рискуете получить зияющую дыру в защите!!!!

В предыдущих версиях этот метод назывался **import()**. Начиная с версии 2.20, это имя было полностью удалено из модуля во избежание конфликта со встроенным в Perl оператором **import**.

УДАЛЕНИЕ ПАРАМЕТРА:

Для того, чтобы удалить параметр, вызовите метод

```
$query->delete('foo');
```

Это иногда бывает полезным для сброса параметров, которые Вы не хотите передавать сценарию при вызове.

Если Вы используете функциональный интерфейс, используйте функцию Delete() чтобы избежать конфликта со встроенным оператором Perl "delete".

УДАЛЕНИЕ ВСЕХ ПАРАМЕТРОВ:

```
$query->delete_all();
```

Этот метод полностью очищает объект CGI. Это может быть полезным, если Вы хотите удостовериться, что переданы только параметры, установленные по умолчанию.

При использовании функционального интерфейса Вам необходимо вызвать функцию `Delete_all()`.

НЕПОСРЕДСТВЕННЫЙ ДОСТУП К СПИСКУ ПАРАМЕТРОВ:

```
$q->param_fetch('address')->[1] = '13-я Парковая улица дом 24';
unshift @{$q->param_fetch(-name=>'address')},'Жора Сергеев';
```

Если Вам нужен доступ к списку параметров способом, который отличается от описанных выше, Вы можете получить непосредственную ссылку на него при помощи вызова метода **param_fetch()** с указанием в нем имени параметра. Это

позволяет возвратить ссылку на массив именованных параметров, которую Вы можете в дальнейшем обрабатывать, как Вам заблагорассудится

Вы также можете пользоваться стилем вызова именованного аргумента, используя аргумент **-name**.

ИЗВЛЕЧЕНИЕ СПИСКА ПАРАМЕТРОВ В ВИДЕ ХЭША:

```
$params = $q->Vars;
print $params->{'address'};
@foo = split("\0",$params->{'foo'});
%params = $q->Vars;
use CGI ':cgi-lib';
$params = Vars;
```

Многие программисты хотят получить полный список параметров в качестве хэша, в котором ключами являются имена параметров CGI, а значениями - значения параметров. Это позволяет сделать метод `Vars()`. Вызванный в скалярном контексте, он возвращает список параметров в виде связанной ссылки на хэш. Изменение ключа изменяет значение параметра в базовом списке параметров CGI. Вызванный в контексте массива, он возвращает список параметров в виде обычного хэша. Это позволяет Вам читать содержимое списка параметров, не изменяя его.

При использовании этого метода основную осторожность нужно проявлять в отношении параметров с множественными значениями. Так как хэш не делает различий между контекстом скаляров и массивов, параметры, имеющие множественные значения, будут возвращены в качестве упакованной строки, использующей в качестве символа - разделителя нулевой символ "\0'' (null). Для того чтобы получить отдельные значения, Вы должны расщепить эту упакованную строку. Это соглашение, представленное давным-давно Стивом Бреннером в его модуле cgi-lib.pl, разработанном для Perl версии 4.

Если Вы хотите использовать `Vars()` в качестве функции, импортируйте набор функций *:cgi-lib* (смотрите также раздел, посвященный совместимости с библиотекой CGI-LIB).

СОХРАНЕНИЕ СОСТОЯНИЯ СЦЕНАРИЯ В ФАЙЛЕ:

```
$query->save(FILEHANDLE)
```

Это предложение запишет текущее состояние формы в файл, имеющий дескриптор FILEHANDLE. Вы сможете прочесть его в дальнейшем, подставив дескриптор в метод `new()`. Обратите внимание на то, что дескриптор может быть файлом, каналом, или чем бы то ни было!

Формат сохраняемого файла следующий:

```
NAME1=VALUE1
NAME1=VALUE1'
NAME2=VALUE2
NAME3=VALUE3
=
```

Как имя, так и значение не содержат в своем теле URL. Параметры CGI, имеющие несколько значений, представляются повторяющимися именами. Запись сессии ограничивается единичным символом равенства =. Вы можете записать несколько сессий, а затем прочесть их при помощи нескольких вызовов метода **new**. Вы можете делать это в течение нескольких сессий, открывая файл в режиме добавления, что позволяет создавать простейшие гостевые книги или хранить в нем историю запросов, пользователя. Ниже приведен короткий пример многосессионной обработки записей:

```
use CGI;
open (OUT,">>test.out") || die;
$records = 5;
foreach (0..$records) {
    my $q = new CGI;
    $q->param(-name=>'counter',-value=>$_);
    $q->save(OUT);
}
close OUT;
# повторное открытие для чтения
open (IN,"test.out") || die;
while (!eof(IN)) {
    my $q = new CGI(IN);
    print $q->param('counter'),"\n";
}
```

Формат файла, используемый для сохранения/восстановления идентичен формату обмена данными Boulderio, и с ним можно работать и даже использовать в качестве базы данных при помощи утилит Boulderio от Whitehead Genome Center, Смотрите http://stein.cshl.org/boulder/ для получения дополнительной информации.

Если Вы хотите использовать данный метод в функционально-ориентированном (не в объектно-ориентированном) контексте, то экспортируемым именем функции для этого метода будет **save_parameters()**.

ИЗВЛЕЧЕНИЕ ОШИБОК CGI

Ошибки могут возникать при обработке информации, вводимой пользователем, особенно при обработке выгруженных файлов. Когда случаются эти ошибки, CGI остановит обработку сценария и вернет пустой список параметров. Вы можете проверить наличие и природу ошибок при помощи функции *cgi_error()*. Сообщения об ошибках отформатированы как коды состояния HTTP. Вы можете либо вставить нужный Вам текст сообщения об ошибке в генерируемую страницу, либо использовать значение кода состояния HTTP:

```
my $error = $q->cgi_error;
if ($error) {
```

```
    print $q->header(-status=>$error),
          $q->start_html('Проблемы...'),
          $q->h2('Запрос не обработан'),
          $q->strong($error);
    exit 0;
}
```

При использовании функционально-ориентированного интерфейса (смотри следующий раздел), ошибки могут возникать только при первом вызове *param()*. Будьте к этому готовы!

ИСПОЛЬЗОВАНИЕ ФУНКЦИОНАЛЬНО - ОРИЕНТИРОВАННОГО ИНТЕРФЕЙСА

Для того, чтобы использовать функционально - ориентированный интерфейс, Вы должны определит, какие процедуры набор процедур CGI.pm нужно импортировать в Ваше пространство имен. При этом подходе появляются небольшие накладные расходы, связанные с процессом импорта, но они не столь уж велики.

```
use CGI <список методов>;
```

Перечисленные методы будут импортированы в текущий пакет, Вы затем можете вызывать их непосредственно, не создавая предварительно объект CGI. Нижеследующий пример показывает как импортировать методы **param()** и **header()** methods, а затем вызывать их непосредственно:

```
use CGI 'param','header';
print header('text/plain');
$zipcode = param('zipcode');
```

Чаще Вы будете импортировать наборы функций, ссылаясь к этой группе по имени. Все имена предопределенных наборов (пакетов) функций начинаются со знака двоеточия ":" , как, например, ":html3" (для работы с набором тэгов, определенных стандартом HTML3).

Ниже приведен список функциональных пакетов, которые Вы можете импортировать:

:cgi
: Импортирует все методы для манипулирования CGI, такие как **param()**, **path_info()** и тому подобные.

:form
: Импортирует все методы для генерирования заполняемых форм, такие как **textfield()**.

:html2
: Импортрует все методы, генерирующие стандартные тэги HTML 2.0.

:html3
: Импортирует все методы, генерирующие набор тэгов, предписанный реализацией HTML 3.0 (такие как <table>, <super> и <sub>).

:netscape

Импортирует все методы, генерирующие расширения HTML, специфичные для браузера Netscape.

:html

Импортирует все методы, генерирующие HTML-тэги (т.е. 'html2' + 'html3' + 'netscape')...

:standard

Импортирует "стандартные'' методы, 'html2', 'html3', 'form' и 'cgi'.

:all

Импортирует все имеющиеся методы. Для получения полного списка смотрите исходный код модуля CGI.pm, а именно, место, где определяется хэш %EXPORT_TAGS.

Если Вы импортируете имя функции, не являющейся компонентом CGI.pm, модуль будет интерпретировать это имя в качестве нового тэга HTML и сгенерирует соответствующую подпрограмму. Вы можете использовать ее в дальнейшем, как и любой другой шаблон-тэг HTML. Эта возможность предусмотрена для того, чтобы обеспечить будущую совместимость со стремительно развивающимися стандартами HTML. Скажем, к примеру, что новый браузер Microsoft ввел новый тэг, названный <GRADIENT> (который вызывает градиентную заливку рабочего стола пользователя в процессе перезагрузки компьютера). Вам нет необходимости ожидать выхода новой версии CGI.pm, а можно начать пользоваться новой возможностью немедленно:

```
use CGI qw/:standard :html3 gradient/;
print gradient({-start=>'red',-end=>'blue'});
```

Заметьте, что в интересах скорости выполнения CGI.pm **не использует** стандартный синтаксис модуля Exporter для явного указания загружаемых символов. Однако это может быть сделано в дальнейшем - смотрите документацию по модулю CGI.pm, установленному в Вашей системе.

Если Вы импортируете любой из структурообразующих или формогенерирующих методов CGI.pm, будет создан и инициализирован объект CGI.pm, предусмотренный по умолчанию, в момент первого же использования любого из таких методов. Список таких методов включает в себя **param()**, **textfield()**, **submit()** и им подобные. (Если Вам потребуется непосредственный доступ к объекту CGI, Вы можете обратиться к глобальной переменной **$CGI::Q**). Путем импортирования методов CGI.pm, Вы можете создавать элегантные привлекательного вида сценарии CGI:

```
use CGI qw/:standard/;
print
   header,
   start_html('Пример сценария'),
   h1('Пример сценария'),
   start_form,
   "Ваше имя? ",textfield('name'),p,
   "Ваши любимые слова?",
   checkbox_group(-name=>'words',
                  -values=>['Хрю','Гав','Мяу','Муу'],
```

```
                -defaults=>['Хрю','Муу']),p,
    "Ваш любимый цвет?",
    popup_menu(-name=>'color',
               -values=>['красный','зеленый','синий','янтарный']),p,
    submit('Грызть!'),
    end_form,
    hr,"\n";
  if (param) {
    print
       "Ваше имя: ",em(param('name')),p,
       "Любимые слова: ",em(join(", ",param('words'))),p,
       "Любимый цвет: ",em(param('color')),".\n";
  }
  print end_html;
```

ДИРЕКТИВЫ ПРЕПРОЦЕССОРА (ПРАГМЫ)

В дополнение к наборам функций имеется целый ряд директив препроцессора, которые Вы также можете импортировать. Эти директивы, которые всегда начинаются с дефиса, изменяют поведение функций модуля CGI.pm. Директивы, наборы функций, отдельные функции могут быть импортированы в одном и том же предложении `use()`. Например, следующее предложение импортирует стандартный набор функций и отключает отладочный режим (директива -no_debug):

```
use CGI qw/:standard -no_debug/;
```

Полный список директив препроцессора приведен далее:

-any

При использовании *CGI*-прагмы ***-any*** любой метод, который не распознан объектом - запросом, будет интерпретирован в качестве нового тэга HTML. Это позволяет Вам будущие расширения стандартного языка HTML, которые предлагаются AOL Netscape или Microsoft. Такая возможность дает Вам ощущение свободы при работе с новыми и не поддерживаемыми тэгами:

```
use CGI qw(-any);
$q=new CGI;
print $q->gradient({speed=>'fast',start=>'red',end=>'blue'});
```

Так как использование *any* вызывает интерпретацию любого ошибочно записанного имени метода в качестве тэга HTML, используйте его внимательно, либо не используйте вообще.

-compile

Эта директива вызывает немедленную компиляцию указанных автоматически загруженных методов, а не откладывает ее на потом. Такая возможность особенно полезна для тех сценариев, которые имеют большое время выполнения в среде FastCGI или mod_perl, а также для тех сценариев, которым предначертано быть перемолотыми компилятором Perl, написанным Малкольмом Битти (Malcom Beatty). Используйте эту директиву в связке с методами или семействами методов, которые Вы планируете использовать.

```
use CGI qw(-compile :standard :html3);
```

или даже (!!!):

```
use CGI qw(-compile :all);
```

Обратите внимание на то, что прагма -compile, использованная таким образом, всегда приводит в результате к импортированию всех скомпилированных функций в текущее пространство имен. Если Вы хотите использовать компиляцию без импортирования, воспользуйтесь вместо этого методом `compile()`, описанным ниже.

-nph

Эта директива заставляет CGI.pm создать заголовок, предназначенный для работы в качестве NPH-сценария (no parsed header - неанализируемые заголовки). Вам может потребоваться выполнить также и некоторые другие действия, а также сообщить серверу, что данный сценарий является NPH-сценарием. Смотрите обсуждение особенностей NPH-сценариев ниже.

-newstyle_urls

Разделяет пары имя=значение в строке передачи параметров CGI-запроса при помощи точек с запятой, а не амперсандов (&). Например:

```
?name=fred;age=24;favorite_color=3
```

Строка запроса, использующая в качестве символа - разделителя точку с запятой, принимается всегда, не может быть выдана в качестве результата использования методова `self_url()` и `query_string()`, до тех пор, пока не использована явно прагма -newstyle_urls.

-autoload

Данная директива отменяет стандартный автозагрузчик, так что любая нераспознанная функция Вашей программы отсылается к CGI.pm для возможного определения. Это позволяет Вам использовать все функции CGI.pm, не добавляя их имена к Вашей таблице символов, что, как правило, очень беспокоит пользователей mod_perl, озабоченных размером оперативной памяти, занятым программой.

Предупреждение: Если действует директива *-autoload*, Вы не можете использовать "свободный стиль" кодирования (запись вызовов функций без явного написания скобок). Используйте *hr()* а не *hr*, или добавьте нечто вроде *use subs qw/hr p header/* в начало Вашего сценария.

-no_debug

Эта директива отключает возможность обработки командной строки. Если Вы хотите запустить сценарий CGI из командной строки для генерации HTML, но при этом не хотите остановки сценария для запроса параметров CGI со стандартного ввода или из командной строки, используйте такую директиву:

```
use CGI qw(-no_debug :standard);
```

Если же Вы хотите обработать параметры командной строки, но не стандартный ввод, то следующая конструкция должна сработать:

```
use CGI qw(-no_debug :standard);
restore_parameters(join('&',@ARGV));
```

Смотрите раздел, посвященный отладке, для получения более детальной информации.

-private_tempfiles

Модуль CGI.pm может обрабатывать выгружаемый файл. Обычно он отправляет выгружаемый файл во временную директорию, а затем удаляет временный файл после полного окончания выгрузки. Однако, это может привести к перехвату данных, как описано в разделе, посвященном выгрузке файла. Автор другого CGI-сценария может намеренно организовать слежение за выгружаемыми данными, особенно, если они представляют конфиденциальную информацию. В Unix-системах директива -private_tempfiles приведет к тому, что временный файл будет разлинкован, как только он открывается и прежде чем в него будут записаны любые данные, уменьшая, но не устраняя полностью риск перехвата (потенциально состязательность все еще остается). Для того, чтобы сделать жизнь хакера тяжелее, программа выбирает имена временных файлов путем вычисления 32-битных контрольных сумм получаенных заголовков HTTP.

Для полной уверенности в том, что не сможет быть прочитан другими сценариями CGI, используйте suEXEC (запуск с правами пользователя-владельца файла) или программу CGI wrapper (sBox) для запуска Вашего сценария. Временный файл создается с атрибутами 0600 (ни группы, ни прочие пользователи не могут читать данный файл).

Временная директория выбирается по следующему алгоритму:

```
1. Если текущий пользователь  (например, "nobody"), имеет в своей
  домашней директории  директорию "tmp",  используется эта директория
  (только в UNIX-системах).
2. Если определена переменная окружения TMPDIR, используется директория,
установленная этой переменной.
3. В противном случае проверяются пути /usr/tmp, /var/tmp, C:\temp,
/tmp, /temp, ::Temporary Items, and \WWW_ROOT.
```

Для каждого из найденных путей проверяется, директория ли это, и имеется ли возможность записи в нее. Если нет, алгоритм пытается использовать следующую альтернативу.

ФОРМА СИНТАКСИСА ДЛЯ ИМПОРТИРОВАНИЯ ОТДЕЛЬНЫХ ФУНКЦИЙ - ТЭГОВ HTML

Многие методы предназначены для генерирования тэгов HTML. Как описывается ниже, тэговые функции генерируют автоматически как открывающие, так и закрывающие тэги. Например, предложение:

```
print h1('Заголовок 1-го уровня');
```

генерирует следующую конструкцию HTML:

```
<H1>Заголовок 1-го уровня</H1>
```

Иногда случаются ситуации, когда Вам нужно сгенерировать по отдельности, открывающие и закрывающие операторные скобки (тэги) HTML. В этом случае Вы можете использовать форму вызова start_*tag_name* и end_*tag_name*, как в нижеследующем примере:

```
print start_h1,'Заголовок уровня 1',end_h1;
```

За некоторыми исключениями (описанными ниже), функции start_*tag_name* и end_*tag_name* не генерируются автоматически при *вызове модуля при помощи use CGI*. Однако, Вы можете указать, какие тэги требуется сгенерировать путем помещения знака звездочки "*" перед именем импортируемой функции, или, в качестве альтернативного синтаксиса, указав "start_*tag_name*'' или "end_*tag_name*" в списке ипортируемых функций.

Пример:

```
use CGI qw/:standard *table start_ul/;
```

В данном примере, в дополнение к стандартным функциям, будут сгенерированы следующие функции:

1. **`start_table()` (генерирует тэг <TABLE>)**
2. **`end_table()` (генерирует тэг </TABLE>)**
3. **`start_ul()` (генерирует тэг <UL>)**
4. **`end_ul()` (генерирует тэг </UL>)**

ГЕНЕРИРОВАНИЕ ДИНАМИЧЕСКИХ ДОКУМЕНТОВ

Большинство функций модуля CGI.pm связано с созданием документов "на лету". Обычно Вы генерируете сначала заголовок HTTP, за которым следует сам документ. CGI.pm обеспечивает функции как для генерации различных типов заголовков HTTP, так и для генерирования HTML. Для генерирования изображений в формате GIF, смотрите модуль GD.pm.

Каждая из этих функций генерирует фрагмент HTML или HTTP, который Вы можете непосредственно вывести в окно браузера, добавить к результирующей строке или сохранить в файле для последующего использования.

СОЗДАНИЕ СТАНДАРТНОГО ЗАГОЛОВКА HTTP:

Обычно первое действие, которое вы выполняете в любом сценарии CGI, - печать заголовка HTTP. Он говорит браузеру, каков тип принимаемого документа, а также предоставляет другую дополнительную информацию, такую как язык, конечная дата хранения, требуется ли кэширование документа. Изменение заголовка может также быть использовано в специальных целях, таких как генерацию страниц, требующих оплату за пользование ресурсом или выталкивание страниц сервером (push-технология).

```
print $query->header;
    -или-
print $query->header('image/gif');
    -или-
print $query->header('text/html','204 Сервер не отвечает');
    -или-
print $query->header(-type=>'image/gif',
                -nph=>1,
                -status=>'402 Требуется оплата за пользование ресурсом',
                -expires=>'+3d',
                -cookie=>$cookie,
                -Cost=>'$2.00');
```

`header()` возвращает заголовок *Content-type:*. Вы можете указать любой выбранный Вами тип MIME, в противном случае будет установлен тип по умолчанию - text/html. Необязательный второй параметр определяет код состояния и диагностическое сообщение, которое может быть прочитано человеком. К примеру, в качестве второго параметра Вы можете указать 204, "Нет ответа сервера'' для создания сценария, который предписывает браузеру вообще НИЧЕГО не делать.

Последний пример показывает стиль передачи именованных аргументов методам CGI при помощи именованных параметров. Распознаются параметры **-type**, **-status**, **-expires**, и **cookie**. В любых других именованных параметрах будет удален начальный знак тире, и они будут превращены в поля заголовка, позволяя таким образом создавать поля заголовков HTTP по Вашему желанию. Внутренние подчеркивания преобразуются в тире:

```
print $query->header(-Content_length=>3002);
```

Большинство браузеров не кэшируют результаты выполнения сценариев CGI. Каждый раз при перезагрузке страницы браузером сценарий вызывается заново. Вы можете изменить такое поведение при помощи параметра **-expires**. Когда Вы определите относительное или абсолютное значение этого интервала - срока действия вывода сценария, некоторые браузеры и прокси-серверы будут кэшировать выводимый сценарием поток до наступления указанной даты. Все нижеприведенные формы указания значения параметра допустимы для поля -expires:

```
+30s                          в течение следующих 30 секунд
+10m                           в течение следующих 10 минут
+1h                           в течение часа
-1d                           вчера (т.е. "БЫСТРО, НАСКОЛЬКО ВОЗМОЖНО!")
now                           немедленно
+3M                           в течение 3-х месяцев
+10y                          в течение 10 лет
Thursday, 25-Apr-2001 06:40:33 GMT  В указанные день и время
```

Параметр **-cookie** генерирует заголовок, который предписывает браузеру передать сценарию файл cookie в течение всех последующих взаимодействий со сценарием. Файлы cookie Netscape имеют специальный формат, который включает в себя некоторые интересные атрибуты, как, например, срок хранения cookie. Используйте метод `cookie()` для создания и получения файлов cookie.

Параметр **-nph**, если установлен в истину, сформирует корректный заголовок для работы со сценариями NPH (no-parse-header - неанализируемые заголовки). Это очень важно для некоторых серверов, таких как Microsoft Internet Information Server (IIS), которые ожидают от своих сценариев, чтобы они выполнялись в режиме NPH.

ГЕНЕРИРОВАНИЕ ЗАГОЛОВКА ПЕРЕНАПРАВЛЕНИЯ (РЕДИРЕКТА)

```
print $query->redirect('http://somewhere.else/in/movie/land');
```

Иногда Вам не нужно генерировать документ HTML, а просто необходимо переадресовать браузер клиента куда-нибудь еще, возможно, на основе времени суток или информации о посетителе.

Функция `redirect()` перенаправляет браузер клиента к другому URL. Если Вы используете такое перенаправление, **ВЫ НЕ ДОЛЖНЫ ПЕЧАТАТЬ** заголовок. Начиная с версии 2.0, модуль генерирует оба типа заголовка - официальный URI: и неофициальный Location:. Это должно удовлетворить большинство серверов и браузеров.

Я могу поделиться с Вами одним полезным наблюдением: относительные ссылки могут работать некорректно, если они ссылаются на документ, размещенный на том же самом сайте. Это происходит потому, что некоторые серверы используют

хорошо продуманную оптимизацию обращений к своим документам. Решением будет использование полного URL (включая начальную часть http://) в адресе документа, на который Вы осуществляете перенаправление.

Вы можете также использовать именованные аргументы:

```
print $query->redirect(-uri=>'http://somewhere.else/in/movie/land',
                       -nph=>1);
```

Параметр **-nph**, если ему присвоено значение 'истина', сгенерирует корректные заголовки для работы со сценариями NPH (no-parse-header). Это очень важно при работе с некоторыми серверами, такими как Microsoft Internet Information Server (IIS), которые подразумевают, что все выполняемые на них сценарии имеют атрибут NPH.

СОЗДАНИЕ ЗАГОЛОВКА ДОКУМЕНТА HTML

```
print $query->start_html(-title=>'Секреты египетских пирамид',
                         -author=>'fred@capricorn.org',
                         -base=>'true',
                         -target=>'_blank',
                         -meta=>{'keywords'=>'фараон секретный мумия',
                                 'copyright'=>' &copy; 1996 Тутанхамон},
                         -style=>{'src'=>'/styles/style1.css'},
                         -BGCOLOR=>'blue');
```

После создания заголовка HTTP, большинство сценариев CGI начинают писать документ HTML. Функция `start_html()` создает заголовок страницы, наряду с большим количеством дополнительной информации, которая управляет отображением страницы и ее поведением.

Этот метод возвращает подготовленный заголовок HTML- страницы и открывающий тэг <BODY>. Все передаваемые параметры - необязательные. При использовании именованных параметров распознаются параметры: -title, -author, -base, -xbase и -target (смотрите объяснение ниже). Любые дополнительные параметры, указанные Вами, такие, как атрибут BGCOLOR, введенный Netscape, дописываются к тэгу <BODY>. Дополнительные параметры должны начинаться со знака "-".

Аргумент **-xbase** позволяет Вам указать операнд HREF для тэга <BASE>, который отправляет относительные ссылки по адресу, отличному от текущего, как в случае

```
-xbase=>"http://home.mcom.com/";
```

Все относительные ссылки будут интерпретированы относительно этого адреса.

Аргумент **-target** позволяет Вам указать имя фрейма по умолчанию, в котором открываются ссылки и генерируемые формы. Смотрите документацию Netscape по использованию фреймов для получения дополнительной информации.

```
-target=>"answer_window"
```

Все относительные ссылки будут интерпретированы относительно этого тэга. Вы можете добавить дополнительную META - информацию к секции <HEADER> документа HTML при помощи аргумента **-meta**. Этот аргумент ожидает ссылку на ассоциативный массив, содержащий пары имя/значение тэгов META. Они будут преобразованы в последовательность тэгов <META> генерируемого документа, которая будет выглядеть примерно так:

```
<META NAME="keywords" CONTENT="фараон секрет мумия">
<META NAME="description" CONTENT="&copy;1996 Фараон Тутанхамон">
```

В нстоящее время не существует поддержки для типа HTTP-EQUIV в тэгах <META>. Это связано с тем, что Вы можете модифицировать заголовок HTTP непосредственно при помощи метода **header()**. Например, если Вы хотите установить значение параметра Refresh в заголовке, сделайте это с использованием метода `header()`:

```
print $q->header(-Refresh=>'10; URL=http://www.capricorn.com');
```

Тэг **-style** используется для внедрения в код генерируемой страницы стилевого оформления (CSS). Дополнительную информацию Вы можете получить в разделе СТИЛЕВОЕ ОФОРМЛЕНИЕ.

Вы можете поместить другие произвольно выбранные элементы HTML в секцию <HEAD> при помощи тэга **-head**. Например, для того чтобы поместить редко используемый элемент <LINK> в секцию <HEAD>, используйте следующую конструкцию:

```
print start_html(-head=>Link({-rel=>'next',
            -href=>'http://www.capricorn.com/s2.html'}));
```

Для внедрения нескольких элементов HTML в секцию <HEAD>, просто передайте в качестве параметра ссылку на массив:

```
print start_html(-head=>[
                    Link({-rel=>'next',
                        -href=>'http://www.capricorn.com/s2.html'}),
                    Link({-rel=>'previous',
                        -href=>'http://www.capricorn.com/s1.html'})
                    ]
            );
```

Для программистов, использующих JAVASCRIPT: Параметры **-script**, **-noScript**, **-onLoad**, **-onMouseOver**, **-onMouseOut** и **-onUnload** используются для того, чтобы организовать вызовы функций JavaScript из Ваших страниц. Параметр **-script** должен указывать на текстовый блок, содержащий определения функций JavaScript function. Этот блок будет помещен внутрь секции <SCRIPT> в заголовке страницы HTML (не в заголовке HTTP). Этот блок размещается в заголовке для того, чтобы загрузить все содержащиеся в теле страницы функции JavaScript в самом начале загрузки, чтобы придать странице функциональность даже в случае, если пользователь прервет загрузку страницы до ее полной

закачки браузером, нажав кнопку "стоп". CGI.pm пытается отформатировать результирующий текст таким образом, что браузеры со встроенной поддержкой JavaScript не смогут "захлебнуться" полученным кодом; к несчастью, имеются браузеры, такие, к примеру, как Chimera для Unix, которые, тем не менее, становятся в тупик при обработке полученного кода.

Параметры **-onLoad** и **-onUnload** указывают на фрагменты кода JavaScript, которые выполняются при открытии и закрытии страницы соответственно Обычно эти параметры представляют собой вызовы функций, которые определены в поле **-script**:

```
$query = new CGI;
print $query->header;
$JSCRIPT=<<END;
// Задать глупый вопрос
function riddle_me_this() {
  var r = prompt("Что передвигается на двух конечностях по утрам, " +
            "на двух конечностях - днем, " +
            "и на трех - по ночам?");
  response(r);
}
// Получить еще более глупый ответ
function response(answer) {
  if (answer == "мужчина")
    alert("А векдь ты угадал, зараза!");
  else
    alert("А вот и не угадал! Попробуй еще раз!");
}
END
print $query->start_html(-title=>'Загадка Сфинкса',
                  -script=>$JSCRIPT);
```

Используйте параметр **-noScript** для передачи некоторого текста HTML тем браузерам, которые не имеют встроенной поддержки JavaScript (или эта поддержка отключена), для того чтобы отобразить некоторую диагностическую информацию.

Netscape 3.0 распознает некоторые атрибуты тэга <SCRIPT>, включая LANGUAGE и SRC. Последнее замечание является особенно интересным, так как позволяет Вам держать код JavaScript в отдельном файле или сценарии CGI, а не захламлять текст каждой страницы исходным кодом. Для того, чтобы воспользоваться этими атрибутами, передайте ссылку на хэш в параметре **-script**, содержащий одну или несколько конструкций -language, -src, или -code:

```
print $q->start_html(-title=>'Загадка Сфинкса',
                -script=>{-language=>'JAVASCRIPT',
                        -src=>'/javascript/sphinx.js'}
                );
print $q->(-title=>'Загадка Сфинкса',
        -script=>{-language=>'PERLSCRIPT',
                -code=>'print "Привет, мир!\n;"'}
        );
```

Последняя особенность позволяет Вам внедрять несколько секций <SCRIPT> в заголовок страницы. Вам требуется только передать секций SCRIPT в качестве

ссылки на массив, - это позволит Вам указать различные исходные файлы для разных диалектов JavaScript. Например:

```
print $q->start_html(-title=>'Загадка Сфинкса',
                     -script=>[
                             { -language => 'JavaScript1.0',
                               -src      => '/javascript/utilities10.js'
                             },
                             { -language => 'JavaScript1.1',
                               -src      => '/javascript/utilities11.js'
                             },
                             { -language => 'JavaScript1.2',
                               -src      => '/javascript/utilities12.js'
                             },
                             { -language => 'JavaScript28.2',
                               -src      => '/javascript/utilities219.js'
                             }
                           ]
                       );
</pre>
```

Если это выглядит некоей маразматической крайностью,прислушайтесь к моему совету и придерживайтесь прямолинейного стиля CGI кодирования.

Смотрите

http://home.netscape.com/eng/mozilla/2.0/handbook/javascript/

для получения более подробной информации о языке JavaScript.

Позиционные старомодные параметры описаны ниже:

Параметры:

- Наименование
- Адрес e-mail автора (сгенерирует тэг <LINK REV ="...">, в случае наличия данного параметра)
- Флаг 'true' если Вы хотите включить тэг <BASE> в заголовок. Это помогает дополнить относительные адреса ссылок до полных абсолютных адресов URL, когда документ меняет адрес, но приводит к тому, что иерархическая древовидная структура, связанная с документом, становится непереносимой. Используйте с осторожностью!
- **, 5, 6...**
  Любые другие параметры, которые Вы хотите включить в тэг <BODY>. Это отличное место для размещения расширений Netscape, таких как цвета и фоновый рисунок (background, text и т.д.)

ЗАВЕРШЕНИЕ ДОКУМЕНТА HTML:

```
print $query->end_html
```

Данная конструкция корректно завершает документ HTML, вставляя тэги </BODY></HTML> в конец формируемого документа.

СОЗДАНИЕ URL ССЫЛАЮЩЕГОСЯ НА СЕБЯ И СОХРАНЯЮЩЕГО ТЕКУЩЕЕ СОСТОЯНИЕ:

```
$myself = $query->self_url;
print "<A HREF=$myself>I'm talking to myself.</A>";
```

Метод `self_url()` возвращает URL, при обращении по которому повторно выполняется данный сценарий, оставляя всю информацию о текущем состоянии неизменной. Это может оказаться полезным, если Вы хотите перепрыгнуть в другое место текущего документа при помощи использования внутренних анкеров - меток внутри одного документа, но не хотите разрушить текущее содержимое формы. Такая задача может быть решена, например, так:

```
$myself = $query->self_url;
print "<A HREF=$myself#table1>Смотри таблицу 1</A>";
print "<A HREF=$myself#table2>Смотри таблицу 2</A>";
print "<A HREF=$myself#yourself>See for yourself</A>";
```

Если Вы хотите в большей степени управлять результатом применения метода, воспользуйтесь вместо этого методом **url()**.

Вы также можете получить необработанную строку запроса при помощи вызова метода query_string():

```
$the_string = $query->query_string;
```

ПОЛУЧЕНИЕ АДРЕСА СЦЕНАРИЯ (URL)

```
$full_url      = $query->url();
$full_url      = $query->url(-full=>1);  #альтернативный синтаксис
$relative_url  = $query->url(-relative=>1);
$absolute_url  = $query->url(-absolute=>1);
$url_with_path = $query->url(-path_info=>1);
$url_with_path_and_query = $query->url(-path_info=>1,-query=>1);
```

url() возвращает URL сценария в разнообразных форматах. Будучи вызванным без параметров, возвращает полную форму URL, включая имя хоста и номер порта

```
http://your.host.com/path/to/script.cgi
```

Вы можете модифицировать формат представления URL при помощи следующих параметров:

-absolute

Если верно, то возвратить абсолютное значение URL, например:
/path/to/script.cgi

-relative

Получить относительное значение URL. Это бывает полезно, если Вы повторно вызываете Ваш сценарий с различными параметрами. Пример:
script.cgi

-full

Возвращает полный URL, точно такой же, как и при вызове сценария без единого аргумента. Этот параметр отменяет действие аргументов -relative и -absolute.

-path (-path_info)

Добавляет дополнительную информацию о пути к URL. Параметр может быть совместно использован с параметрами **-full**, **-absolute** или **-relative**. В качестве синонима может быть использовано также имя **-path_info**.

-query (-query_string)

Добавляет строку запроса к URL. Параметр может быть совместно использован с параметрами **-full**, **-absolute** или **-relative**. В качестве синонима может быть использовано имя **-query_string**.

СМЕШИВАНИЕ ПАРАМЕТРОВ ПЕРЕДАННЫХ МЕТОДОМ POST И ПЕРЕДАВАЕМЫХ ЧЕРЕЗ URL

```
$color = $query->url_param('color');
```

Для сценария имеется возможность получить параметры CGI в строке URL как непосредственно, так и при помощи заполненной формы, которая формирует строку URL, содержащую запрос (знак вопроса "?" за которым следуют аргументы). Метод **param()** всегда возвращает содержимое заполненной формы, переданной методом POST, игнорируя строку запроса, переданную через URL. Для того, чтобы получить параметры из URL, вызовите метод **url_param()**. Он используется точно таким же образом, как и метод **param()**. Главное различие заключается в том, что он позволяет Вам читать параметры, но не устанавливать их.

Ни при каких обстоятельствах не называйте параметры, переданные в строке URL, именами, присвоенными параметрам, переданным при помощи метода POST. Если Вы попытаетесь назвать параметры, переданные в строке URL, идентично именам, переданным при помощи метода GET, результаты будут для вас неожиданными.

СОЗДАНИЕ СТАНДАРТНЫХ ЭЛЕМЕНТОВ HTML:

Модуль CGI.pm определяет стандартные методы-шаблоны (сокращения) для генерации большинства, если не для всех тэгов HTML стандартов HTML3 и HTML4. Методы - сокращения HTML названы по именам уникальных элементов HTML и возвращают фрагмент текста HTML, который Вы в дальнейшем можете печатать и манипулировать им, как Вам вздумается. Каждый метод - сокращение возвращает фрагмент кода HTML, который Вы можете добавить к строке, сохранить в файле, или, что случается чаще всего, напечатать его в выходной поток для отображения в окне браузера.

Этот пример показывает, как пользоваться методами-сокращениями HTML:

```
$q = new CGI;
print $q->blockquote(
                "Давным-давно на острове ",
                $q->a({href=>"http://crete.org/";},"Крит"),
                "жил Минотавр по имени ",
                $q->strong("Федя."),
               ),
    $q->hr;
```

Этот фрагмент генерирует следующий код HTML (мы добавили дополнительные символы новой строки для повышения читаемости):

```
<blockquote>
Давным-давно на острове <a HREF="http://crete.org/";>Крит</a>
жил Минотавр по имени <strong>Федя.</strong>
</blockquote>
<hr>
```

Если Вы находите объектно-ориентированный синтаксис вызова методов - сокращений HTML неудобным, Вы можете импортировать их в свое пространство имен и полностью освободиться от объектного синтаксиса (подробная информация приведена в следующем разделе):

```
use CGI ':standard';
print blockquote(
   "Много лет тому назад на острове ",
   a({href=>"http://crete.org/";},"Крит"),
   "жил минотавр по Имени",
   strong("Федя."),
   ),
   hr;
```

ПЕРЕДАЧА АРГУМЕНТОВ МЕТОДАМ - СОКРАЩЕНИЯМ HTML

Методам - сокращениям HTML можно передать один, несколько или ни одного аргумента. Если Вы не передаете аргументы методу, то в результате получаете отдельный тэг:

```
print hr;    #  <HR>
```

Если Вы передаете методу один или несколько строковых аргументов, они сцепляются между собой при помощи пробела и помещаются между открывающим и закрывающим тэгами:

```
print h1("Глава","1"); # <H1>Глава 1</H1>"
```

Если первый аргумент представляет собой ссылку на ассоциативный массив, ключи и значения этого ассоциативного массива становятся атрибутами тэга HTML:

```
print a({-href=>'fred.html',-target=>'_new'},
  "Открыть новый фрейм");
      <A HREF="fred.html",TARGET="_new">Открыть новый фрейм</A>
```

Вы можете и не использовать знак "тире" перед именами атрибутов, если Вам так нравится:

```
print img {src=>'fred.gif',align=>'LEFT'};
      <IMG ALIGN="LEFT" SRC="fred.gif">
```

Иногда тэг HTML не имеет аргументов. Например, пронумерованный список может быть помечен как COMPACT. Синтаксис такого вызова - это аргумент , который указывает на строку undef:

```
print ol({compact=>undef},li('один'),li('два'),li('три'));
```

В версиях CGI.pm, предшествовавших 2.41, передача пустой ('') строки в качестве аргумента атрибута имела эффект, идентичный указанию undef. Однако это соглашение было изменено, чтобы корректно сгенерировать код тэга <IMG ALT="">. Два примера кода, приведенных ниже, демонстрируют различие:

```
КОД                 РЕЗУЛЬТАТ
img({alt=>undef})     <IMG ALT>
img({alt=>''})        <IMG ALT="">
```

НАСЛЕДОВАНИЕ СВОЙСТВ ШАБЛОННЫХ ФУНКЦИЙ HTML

Одна из самых крутых возможностей шаблонных функций HTML - это их способность к распространению свойств. Если Вы передаете им аргумент -

ссылку на список, тэг применяется к КАЖДОМУ элементу списка. Например, вот так можно создать ненумерованный HTML - список:

```
print ul(
        li({-type=>'disc'},['Несерьезный','Умный','Сонный','Счастливый']);
       );
```

Этот пример генерирует в результате код HTML, который выглядит примерно так:

```
<UL>
  <LI TYPE="disc">Несерьезный</LI>
  <LI TYPE="disc">Умный</LI>
  <LI TYPE="disc">Сонный</LI>
  <LI TYPE="disc">Счастливый</LI>
</UL>
```

Это исключительно полезно при создании таблиц. Например:

```
print table({-border=>undef},
       caption('Когда Вы должны съесть свои овощи?'),
       Tr({-align=>CENTER,-valign=>TOP},
       [
          th(['Овощи', 'Завтрак','Ленч','Обед']),
          td(['Помидоры' , 'нет', 'да', 'да']),
          td(['Броколи' , 'нет', 'нет',  'да']),
          td(['Лук'    , 'да','да', 'да'])
       ]
       )
    );
```

МЕТОДЫ - СОКРАЩЕНИЯ HTML И ВСТАВКА РАЗДЕЛИТЕЛЕЙ МЕЖДУ ЭЛЕМЕНТАМИ СПИСКОВ

Рассмотрим этот элемент кода:

```
print blockquote(em('Привет,'),'мамочка!'));
```

Он обычно возвращает ожидаемую Вами строку, а именно:

```
<BLOCKQUOTE><EM>Привет,</EM>мамочка!</BLOCKQUOTE>
```

Обратите внимание на пробел между элементами Привет," и "мамочка!". Модуль CGI.pm вставляет дополнительный пробел при помощи механизма интерполяции массивов, который управляется таинственной переменной $". Иногда дополнительный пробел - это не то, что Вы хотите получить, к примеру, если перед Вами стоит задача - выровнять серию рисунков. В этом случае Вы можете присвоить переменной $" пустую строку.

```
{
  local($") = '';
  print blockquote(em('Привет,'),'мамочка!'));
 }
```

Я полагаю, что Вы поместите этот элемент кода в блок, как показано выше, в противном случае изменение переменной $" повлияет на весь последующий код сценария, до тех пор, пока Вы явно не измените ее значение.

НЕСТАНДАРТНЫЕ ШАБЛОННЫЕ ФУНКЦИИ HTML

Некоторые шаблонные функции HTML по различным причинам не следуют стандартным соглашениям.

comment() генерирует комментарий HTML (<!-- комментарий -->). Ее можно вызвать так:

```
print comment('Это мой комментарий');
```

Из-за конфликта со встроенными функциями Perl ,следующие функии CGI.pm пишутся с большой буквы:

```
Select
Tr
Link
Delete
Accept
Sub
```

Вдобавок, `start_html(), end_html(), start_form(), end_form(), start_multipart_form()` и все тэги заполнения форм имеют свои особенности. Смотрите соответствующие разделы настоящего документа.

КРАСИВО ОТФОРМАТИРОВАННЫЙ ТЕКСТ HTML

По умолчанию, текст HTML сгенерированный этими функциями, представляет собой одну длинную строку, не отформатированную при помощи символов перевода строки или табуляции (отступов). Это некрасиво, но в результате уменьшает размер документа на 10-20%. Для получения красиво отформатированного документа используйте дочерний класс CGI::Pretty, разработанный и предложенный Брайаном Паулсеном (Brian Paulsen).

СОЗДАНИЕ ФОРМ:

Общее примечание: Все имеющиеся методы, генерирующие элементы формы, возвращают в вызывающую программу строки, содержащие тэг (тэги), создающие требуемый элемент формы. Вы отвечаете за реальный вывод (печать в поток) этих строк. Это сделано именно так, для того, чтобы Вы могли поместить тэги форматирования вокруг тэгов - элементов формы.

Еще одно примечание: Значения по умолчанию, которые Вы устанавливаете для элементов формы, используются только **при первом вызове сценария** (когда еще нет строки запроса). При последующих вызовах сценария (когда сформирована строка запроса), используются последние установленные значения, даже если они представляют собой пустые строки.

Вы можете изменить предыдущее значение поля двумя способами:

(1) вызовите метод `param()` для того, чтобы установить нужное значение.

(2) используйте параметр -override (синоним: -force), - новую возможность, введенную в версии 2.15. Это приводит к использованию значения, установленного по умолчанию, вне зависимости от предыдущего значения:

```
print $query->textfield(-name=>'field_name',
                        -default=>'начальное значение',
                        -override=>1,
                        -size=>50,
                        -maxlength=>80);
```

И еще одно примечание: По умолчанию, текст и подписи к элементам формы используют esc-полследовательности для кодирования спецсимволов. Это означает, что Вы можете безопасно использовать наименования типа "<НАЖМИ МЕНЯ>" в качестве надписи на кнопке. Однако, это влияет на возможность ввода в поля специальных HTML - esc - последовательностей, таких как Á. Если Вы хотите отключить автоматическое преобразование символов в esc - последовательности, вызовите метод `autoEscape()` с параметром "ложь" сразу же после создания объекта CGI:

```
$query = new CGI;
$query->autoEscape(undef);
```

СОЗДАНИЕ ТЭГА ISINDEX

```
print $query->isindex(-action=>$action);
     -или-
print $query->isindex($action);
```

Печатает очень древний тэг <ISINDEX>. Не очень впечатляющая конструкция, особенно, если учесть, что в стандарте HTML 4 этот тэг уже отсутствует.

Параметр -action определяет URL сценария, обрабатывающего данный запрос. По умолчанию запрос обрабатывается текущим сценарием.

НАЧАЛО И ЗАВЕРШЕНИЕ КОДА ФОРМЫ

```
print $query->startform(-method=>$method,
                        -action=>$action,
                        -enctype=>$encoding);
  <... различные данные формы ...>
print $query->endform;
    -или-
print $query->startform($method,$action,$encoding);
  <... различные данные формы ...>
print $query->endform;
```

Вызов `startform()` вернет тэг <FORM> с необязательными параметрами, которые Вы укажете: метод, действие и способ кодирования, использованный для передачи параметров. Значения по умолчанию:

```
method: POST
action: this script
enctype: application/x-www-form-urlencoded
```

Метод `endform()` возвращает закрывающий тэг </FORM>.

Аргумент enctype метода startform() указывает браузеру, как упаковать различные поля формы перед отправкой их серверу, на котором выполняется сценарий. Возможны следующие значения аргумента enctype:

application/x-www-form-urlencoded
: Это старый тип кодирования, используемый всеми браузерами - предшественниками Netscape 2.0. Он совместим со многими сценариями CGI и пригоден для коротких полей, содержащих текстовые данные. Для Вашего удобства, CGI.pm хранит имя этого метода кодирования в переменной **$CGI::URL_ENCODED**.

multipart/form-data
: Это новая форма кодирования, которая начала применяться с выходом Netscape версии 2.0. Она пригодна для форм, содержащих поля очень большого размера или которые предназначены для передачи двоичных данных. Наиболее важным следствием введения этого способа является возможность "выгрузки файла", введенная в формах Netscape 2.0. Для Вашего удобства, CGI.pm сохраняет имя этого типа кодирования в переменной **&CGI::MULTIPART**

  Формы, которые используют этот тип кодирования, достаточно тяжело обрабатывались сценариями CGI, пока не были разработаны библиотеки и модули, такие, как CGI.pm, в которые включены соответствующие возможности.

В целях совместимости метод `startform()` использует старую форму кодирования по умолчанию. Если Вы хотите использовать новую форму кодирования вместо

установленной по умолчанию, вызовите метод **start_multipart_form()** вместо **startform()**.

Для программистов, использующих JAVASCRIPT: Для использования в JavaScript предусмотрены параметры **-name** и **-onSubmit**. Параметр -name присваивает форме имя, что дает возможность функциям JavaScript обрабатывать форму и отличать ее от других форм. Параметр -onSubmit должен указывать на функцию JavaScript которая выполняется ***до*** того как форма будет передана Вашему серверу Вы можете использовать эту возможность для проверки содержимого формы на правильность ввода и полноту заполнения полей. Если Вы обнаружили что-либо неверное с Вашей точки зрения, Вы можете отобразить окно с предупредительным сообщением или исправить что-либо самостоятельно. Вы также можете отменить отправку запроса на сервер, вернув в качестве ответа функции значение "ложь".

Обычно весь набор функций JavaScript определяется в блоке <SCRIPT> в заголовке страницы HTML секции <HEADER> и параметр -onSubmit указывает на одну из этих функций. Смотрите `start_html()` для получения детальной информации.

СОЗДАНИЕ ТЕКСТОВОГО ПОЛЯ

```
print $query->textfield(-name=>'field_name',
                        -default=>'начальное значение',
                        -size=>50,
                        -maxlength=>80);
   -или-
print $query->textfield('field_name','начальное значение',50,80);
```

Метод `textfield()` возвращает текстовое поле ввода.

Параметры

- Первый параметр (требуется) - имя поля (-name).
- Необязательный второй параметр - начальное значение по умолчанию для поля (-default).
- Необязательный третий параметр - размер поля в символах (-size).
- Необязательный четвертый параметр - максимальное количество символов, которое можно ввести в поле (-maxlength).

Что касается всех этих методов, поле инициализируется его прежним содержимым, взятым из предыдущего вызова сценария. При обработке формы значение текстового поля может быть получено при помощи конструкции:

```
$value = $query->param('foo');
```

После вызова сценария Вы можете установить другое значение параметра при помощи следующей конструкции:

```
$query->param('foo',"Я принимаю это значение!");
```

НОВОЕ В ВЕРСИИ 2.15: Если Вы не хотите, чтобы поле принимало свое прежнее значение, Вы можете заставить его принять текущее значение при помощи параметра -override (синоним: -force):

```
print $query->textfield(-name=>'field_name',
                        -default=>'начальное значение',
                        -override=>1,
                        -size=>50,
                        -maxlength=>80);
```

Для программистов, использующих JAVASCRIPT: Вы можете указать также параметры **-onChange**, **-onFocus**, **-onBlur**, **-onMouseOver**, **-onMouseOut** и **-onSelect** для описания обработчков событий JavaScript. Обработчик onChange бдет вызван, если пользователь изменит содержимое текстового поля. По Вашему желанию Вы можете выполнить проверку введенного значения. onFocus и onBlur соответственно вызываются, когда фокус ввода перемещается или уходит из текстового поля ввода. onSelect вызывается, когда пользователь изменяет выбранную часть текста.

СОЗДАНИЕ МНОГОСТРОЧНОГО ТЕКСТОВОГО ПОЛЯ

```
print $query->textarea(-name=>'foo',
                       -default=>'начальное значение',
                       -rows=>10,
                       -columns=>50);
    -или-
print $query->textarea('foo','начальное значение',10,50);
```

Модуль `textarea()` ведет себя подобно обычному методу, реализующему текстовое поле ввода, однако позволяет указать количество строк и текстовых позиций для многострочного окна ввода текста. Вы также можете указать начальное значение поля, которое может быть длинным и состоять из нескольких строк.

Для программистов, использующих JAVASCRIPT: Распознаются параметры **-onChange**, **-onFocus**, **-onBlur** , **-onMouseOver**, **-onMouseOut**, и **-onSelect**. Смотрите описание `textfield()`.

СОЗДАНИЕ ПАРОЛЬНОГО ПОЛЯ

```
print $query->password_field(-name=>'secret',
                             -value=>'начальное значение',
                             -size=>50,
                             -maxlength=>80);
    -или-
print $query->password_field('secret','начальное значение',50,80);
```

Метод `password_field()` действует подобно методу textfield(), за исключением того, что его содержимое заменяется звездочками при отображении на странице.

Для программистов, использующих JAVASCRIPT: Распознаются параметры **-onChange**, **-onFocus**, **-onBlur**, **-onMouseOver**, **-onMouseOut** и **-onSelect** . См. `textfield()`.

СОЗДАНИЕ ПОЛЯ ВЫГРУЗКИ ФАЙЛА

```
print $query->filefield(-name=>'uploaded_file',
                        -default=>'начальное значение',
                        -size=>50,
                        -maxlength=>80);
   -или-
print $query->filefield('uploaded_file','начальное значение',50,80);
```

Функция `filefield()` возвращает поле выгрузки файлов для браузеров, совместимых с версией Netscape 2.0+. Чтобы получить полную отдачу от использования этой функции, Вы обязаны использовать новую схему построения формы - *многосекционную кодированную схему формы* (**multipart_form**). Это можно сделать, вызвав метод **startform()** и указав в качестве типа кодирования **$CGI::MULTIPART**, или путем вызова нового метода **start_multipart_form()** вместо рафинированного старого доброго **startform()**.

Параметры

- Первый обязательный параметр - имя поля (-name).
- Второй необязательный параметр - начальное значение, помещаемое в поле по умолчанию, которое будет использоваться в качестве имени файла (-default).

  В целях обеспечения секретности, браузер не должен привлекать внимания к этому полю, следовательно, начальное значение всего должно быть пустым. Хуже всего то, что поле теряет свою способность "помнить" предыдущее значение. Однако, начальное значение поля требуется по спецификации HTML и, возможно, какой-нибудь браузер, в конечном счете, будет обеспечивать поддержку присвоения значения такому полю.

- Третий необязательный параметр - размер текстового поля в символах (-size).
- Необязательный четвертый параметр - максимальное количество символов, которое можно ввести в поле (-maxlength).

При обработке формы Вы можете получить введенное имя файла путем вызова param():

```
$filename = $query->param('uploaded_file');
```

Различные браузеры возвращают слегка отличающиеся значения имени. Некоторые браузеры возвращают только имя файла. Другие возвращают имя, включающее полный путь к файлу, используя соглашения о написании пути, принятые на компьютере. Вне зависимости от вышесказанного, возвращаемое имя - это имя файла на компьютере пользователя, и не имеет никакого

отношения к временному файлу, который CGI.pm создает в процессе выгрузки (смотрите далее).

Возвращаемое имя файла служит также и дескриптором файла. Вы затем можете читать содержимое этого файла, используя стандартные вызовы процедур Perl для обработки файлов:

```
# Прочесть и напечатать в выходной поток текстовый файл
while (<$filename>) {
   print;
}
# Копировать двоичный файл в дргое место безопасным способом
open (OUTFILE,">>/usr/local/web/users/feedback");
while ($bytesread=read($filename,$buffer,1024)) {
   print OUTFILE $buffer;
}
```

Однако, имеются проблемы, связанные с двойственной природой полей выгрузки файлов. Если Вы используете директиву `use strict`, то Perl возмутится, когда Вы попытаетесь использовать строку в качестве дескриптора файла. Вы можете обойти это предупреждение, поместив код чтения файла в блок, содержащий прагму `no strict`. Более серьезной проблемой будет ввод пользователем "мусора" в поле выгрузки файла. В этом случае результат применения метода `param()` будет не дескриптором файла, а обычной строкой.

Для того чтобы обеспечить безопасность, пользуйтесь функцией *upload()* - новой функцией, введенной в версии 2.47. При вызове с именем файла в качестве поля выгрузки файла, функция *upload()* возвращает дескриптор файла, или undef, если параметр не является допустимым дескриптором.

```
$fh = $query->upload('uploaded_file');
while (<$fh>) {
     print;
}
```

Это - рекомендуемый стиль.

При выгрузке файла браузер обычно посылает некоторую дополнительную информацию вместе с файлом в заголовке HTTP. Эта информация обычно включает в себя тип MIME, приписываемый передаваемому содержимому. Будущие браузеры смогут также посылать и другую информацию (такую, как, например, дату изменения и размер файла). Для того, чтобы получить эту информацию, вызовите метод uploadInfo(). Он возвращает ссылку на ассоциативный массив, содержащий все заголовки документа.

```
$filename = $query->param('uploaded_file');
$type = $query->uploadInfo($filename)->{'Content-Type'};
unless ($type eq 'text/html') {
   die "Только файлы HTML!";
}
```

Если Вы используете машину, которая распознает "текстовый" и "двоичный" режимы передачи данных, убедитесь, что Вы понимаете, когда и как их

использовать (смотрите книгу с Верблюдом - Рэндала Шварца и Тома Кристиансена - the Camel book). В противном случае Вы можете внезапно обнаружить, что двоичные файлы разрушены в процессе выгрузки

Время от времени возникают проблемы, связанные с анализом выгруженного пользователем. Это обычно случается , когда пользователь нажимает кнопку "стоп" до момента окончания выгрузки. В этом случае модуль CGI.pm возвратит значение undef в качестве имени выгруженного файла и передаст методу *cgi_error()* строку "400 Bad request (malformed multipart POST)'' - 400 - Плохой запрос. (Плохо сформированный многосекционный пакет) Это сообщение об ошибке разработано таким, чтобы Вы могли вставить его в код завершения процесса для отправки браузеру. Например:

```
$file = $query->upload('uploaded_file');
if (!$file && $query->cgi_error) {
   print $query->header(-status=>$query->cgi_error);
   exit 0;
}
```

Вы можете по желанию создать собственную страницу, сообщающую об ошибке.

Примечание для программистов, использующих JAVASCRIPT: Распознаются параметры **-onChange**, **-onFocus**, **-onBlur**, **-onMouseOver**, **-onMouseOut** и **-onSelect**. Смотрите подробное описание в разделе, посвященном методу `textfield()`.

СОЗДАНИЕ ВЫПАДАЮЩЕГО (POPUP) МЕНЮ

```
print $query->popup_menu('menu_name',
                         ['Хрю','Гав','Мяу'],
                         'Гав');
  -или-
%labels = ('eenie'=>'Первая альтернатива',
           'meenie'=>'Вторая альтернатива',
           'minie'=>'Третья альтернатива');
print $query->popup_menu('menu_name',
                         ['eenie','meenie','minie'],
                         'meenie',\%labels);
    -или  (стиль передачи именованных параметров)-
print $query->popup_menu(-name=>'menu_name',
                         -values=>['eenie','meenie','minie'],
                         -default=>'meenie',
                         -labels=>\%labels);
```

Метод `popup_menu()` создает выпадающее меню (popup).

1. Обязательный первый аргумент - имя меню (-name).
2. Обязательный второй аргумент (-values) это **ссылка** на массив, содержащий список элементов выпадающего меню. Вы можете передать методу анонимный массив, как в предыдущих примерах, или ссылку на именованный массив, такую, как "\@foo".

3. Необязательный третий параметр (-default) - имя выбранного по умолчанию пункта меню. Если параметр не задан, в качестве параметра по умолчанию будет принят первый пункт меню. Значение предыдущего выбора будут сохранены и переданы запросу при следующем выполнении сценария
4. Необязательный четвертый параметр (-labels) специально разработан для тех программистов, которые хотят использовать в качестве отображаемых элементов выпадающего меню значения, отличающиеся от значений, возвращаемых Вашему сценарию. Он представляет собой указатель на ассоциативный массив соответствия элементов меню отображаемым элементам. Если вы оставите этот параметр пустым, по умолчанию будут отображены значения элементов меню. (Вы можете даже оставить этот параметр неопределенным - undef, - если захотите).

При обработке формы выбранное значение из выпадающего меню может быть получено при помощи:

```
$popup_menu_value = $query->param('menu_name');
```

Примечание для программистов, использующих JAVASCRIPT: Метод `popup_menu()` распознает следующие обработчики событий: **-onChange**, **-onFocus**, **-onMouseOver**, **-onMouseOut** и **-onBlur**. Смотрите детальное описание вызова обработчиков в разделе, посвященном методу `textfield()`.

СОЗДАНИЕ ПРОКРУЧИВАЕМОГО СПИСКА С ВОЗМОЖНОСТЬЮ МНОЖЕСТВЕННОГО ВЫБОРА

```
print $query->scrolling_list('list_name',
                             ['Бее','Мяу','Гав','Муу'],
                             ['Бее','Муу'],5,'true');
  -или-
print $query->scrolling_list('list_name',
                             ['Бее','Мяу','Гав','Муу'],
                             ['Бее','Муу'],5,'true',
                             \%labels);
    -или-
print $query->scrolling_list(-name=>'list_name',
                             -values=>['Бее','Мяу','Гав','Муу'],
                             -default=>['Бее','Муу'],
                             -size=>5,
                             -multiple=>'true',
                             -labels=>\%labels);
```

Метод `scrolling_list()` создает прокручиваемый список с возможностью множественного выбора

Параметры:

- Первый и второй аргументы - это имя списка (-name) и выводимые значения (-values). Как и в случае выпадающего меню, второй аргумент должен быть ссылкой на массив.
- Необязательный третий аргумент (-default) может быть либо ссылкой на список, содержащий значения, выбранные по умолчанию, либо содержать

одиночное выбранное значение. Если аргумент опущен либо не определен, то при первом отображении списка ни один из элементов списка не будет выбран. В случае использования именованного параметра Вы можете также использовать для этого параметра синоним "-defaults".

- Необязательный четвертый параметр - размер списка (-size).
- Необязательный пятый аргумент может быть установлен в 'true', чтобы разрешить одновременный выбор нескольких элементов (-multiple). В противном случае пользователь может выбрать только одно значение из списка.
- Необязательный шестой параметр (-labels) - это указатель на ассоциативный массив, содержащий длинные имена, которые отображаются в качестве надписей, присвоенных элементам списка, видимым пользователю. Если параметр не установлен, будут отображены по умолчанию значения списка в качестве видимых надписей.

При обработке сформированного запроса, все выбранные элементы списка возвращаются в виде массива при опросе параметра с именем 'list_name'. Таким образом можно получить значения, выбранные пользователем из списка:

```
@selected = $query->param('list_name');
```

Примечание для программистов, использующих JAVASCRIPT: Метод `scrolling_list()` распознает следующие обработчики событий: **-onChange**, **-onFocus**, **-onMouseOver**, **-onMouseOut** и **-onBlur**. Смотрите описание метода `textfield()` о способах вызова этих обработчиков

СОЗДАНИЕ ГРУППЫ СВЯЗАННЫХ ПОЛЕЙ-ФЛАЖКОВ

```
print $query->checkbox_group(-name=>'group_name',
                             -values=>['Бее','Мяу','Гав','Муу'],
                             -default=>['Бее','Муу'],
                             -linebreak=>'true',
                             -labels=>\%labels);
print $query->checkbox_group('group_name',
                             ['Бее','Мяу','Гав','Муу'],
                             ['Бее','Муу'],'true',\%labels);
 Только для HTML3-совместимых браузеров:
print $query->checkbox_group(-name=>'group_name',
                             -values=>['Бее','Мяу','Гав','Муу'],
                             -rows=2,-columns=>2);
```

Метод `checkbox_group()` создает группу связанных полей-флажков, объединенных одним и тем же именем.

Параметры:

- Первый и второй аргументы - это имя (-name) группы флажков и их значения (-values) соответственно. Как и в случае с выпадающим меню, второй аргумент должен быть ссылкой на массив. Эти значения

используются в качестве надписей - меток, которые печатаются вслед за полем - флажком, а также для передачи значений сценарию в строке запроса..

- Необязательный третий аргумент (-default) может быть либо ссылкой на список, содержащий значения, отмеченные по умолчанию, либо одиночным отмеченным значением из списка. Если этот аргумент отсутствует или не определен, то при первом появлении списка связанных флажков ни один из них не будет отмечен.
- Необязательный четвертый параметр (-linebreak) может быть установлен в 'true' для того, чтобы поместить символы перевода строки между полями - флажками, для того чтобы отобразить список вертикально, В противном случае поля - флажки с метками - надписями будут расположены в линию - по горизонтали.
- Необязательный пятый аргумент - это указатель на ассоциативный массив соответствия значений полей - флажков подписям, видимым пользователем, которые появляются следом за полями (-labels). Если аргумент не использован, по умолчанию используются значения флажков.
- **HTML3-совместимые браузеры** (такие как Netscape) могут обработать дополнительные необязательные параметры **-rows** и **-columns**. Эти параметры заставляют метод `checkbox_group()` сформировать HTML3 - совместимую таблицу, содержащую группу связанных полей - флажков, отформатированную в соответствии с указанным количеством строк и столбцов. Вы можете указать только параметр -columns если захотите; метод `checkbox_group` вычислит необходимое количество строк в формируемой таблице автоматически.

Для включения заголовков колонок и строк в результирующую таблицу, Вы можете использовать параметры **-rowheaders** и **-colheaders**. Для использования этих параметров им нужно присвоить указатель на массив заголовков. Заголовки несут на себе чисто декоративную функцию. Они не переопределяют поведение группы связанных флажков - это просто именованные величины.

Когда форма обрабатывается сервером, все выбранные (отмеченные) поля будут возвращены в качестве списка при проверке параметра 'group_name'. Значения отмеченных ("on") флажков может быть получены при помощи вызова метода `param()`:

```
@turned_on = $query->param('group_name');
```

Значение, возвращаемое методом `checkbox_group()` представляет собой массив элементов списка кнопок. Вы можете присвоить его массиву, а затем использовать его внутри таблиц, списков или в любых других созидательных целях.:

```
@h = $query->checkbox_group(-name=>'group_name',-values=>\@values);
&use_in_creative_way(@h);
```

Примечание для программистов, использующих JAVASCRIPT: Метод `checkbox_group()` распознает параметр **-onClick**. Он определяет фрагмент кода JavaScript или вызов функции, которая должна выполняться каждый раз, когда пользователь щелкает мышью на любой из кнопок группы. Вы можете получить информацию о том, какая кнопка была нажата при помощи переменной "this".

СОЗДАНИЕ ПОЛЯ-ФЛАЖКА (CHECKBOX)

```
print $query->checkbox(-name=>'checkbox_name',
                       -checked=>'checked',
                       -value=>'ON',
                       -label=>'Отметь меня!');
   -или-
print $query->checkbox('checkbox_name','checked','ON','Выбери меня!');
```

Метод `checkbox()` используется для создания отдельно стоящего поля - флажка, который логически не связан с другими подобными полями.

Параметры:

- Обязательный первый параметр - это имя флажка (-name). Он также будет использован в качестве метки, которая выводится следом за полем-флажком.
- Необязательный второй параметр (-checked) определяет, отмечен ли флажок по умолчанию. Значения, которые могут быть присвоены параметру: -selected и -on (синонимы).
- Необязательный третий параметр (-value) определяет значение флажка, когда он отмечен. Если параметр не указан, по умолчанию подразумевается слово "on".
- Необязательный четвертый параметр (-label) - это подпись, следующая за полем-флажком. Если параметр не определен, по умолчанию используется метка, совпадающая с именем поля-флажка.

Значение флажка может быть получено при помощи:

```
$turned_on = $query->param('checkbox_name');
```

Примечание для использующих JAVASCRIPT: Метод `checkbox()` распознает параметр **-onClick**. Смотрите описание метода `checkbox_group()` для получения более детальной информации.

СОЗДАНИЕ ГРУППЫ РАДИОКНОПОК

```
print $query->radio_group(-name=>'group_name',
                          -values=>['Мяу','Гав','Муу'],
                          -default=>'Гав',
                          -linebreak=>'true',
                          -labels=>\%labels);
   -или-
print $query->radio_group('group_name',['Мяу','Гав','Муу'],
                          'Гав','true',\%labels);
```

Только для HTML-3 - совместимых браузеров:

```
print $query->radio_group(-name=>'group_name',
                          -values=>['Бее','Мяу','Гав','Муу'],
                          -rows=2,-columns=>2);
```

Метод `radio_group()` создает набор логически связанных радиокнопок (выбор только одного значения из группы связанных значений - включает только одну кнопку из набора и выключает остальные)

Параметры:

- Первый обязательный аргумент - имя группы (-name).
- Второй аргумент (-values) - список значений для радиокнопок. Значения идентичны меткам кнопок, которые отображаются на странице. В качестве передаваемого значения аргумента необходимо указать *ссылку* на массив - либо анонимный, как показано в примере выше, либо именованный, например "\@foo".
- Необязательный третий параметр (-default) - имя кнопки, которая по умолчанию включена. Если параметр не указан, первая кнопка будет выбрана по умолчанию. Для того, чтобы сначала ни одна кнопка не была выбрана, укажите в качестве значения этого параметра несуществующее имя кнопки, например, начинающееся со знака "-".
- Необязательный четвертый параметр (-linebreak) которому можно присвоить значение 'true' для того, чтобы вставить между кнопками символы перевода строки, создав, таким образом, вертикальный список кнопок.
- Необязательный четвертый параметр (-labels) - это указатель на ассоциативный массив, связывающий значения радиокнопок с метками кнопок, отображаемыми в окне браузера. Если параметр не указан, то в качестве меток кнопок отображаются сами значения.
- **HTML3-совместимые браузеры** (такие как Netscape) имеют дополнительную функциональность за счет использования дополнительных необязательных параметров **-rows** и **-columns**. Действие этих параметров заставляет `radio_group()` сгенерировать HTML3 - совместимую таблицу, содержащую группу радиокнопок, отформатированную в соответствие с указанным количеством строк и столбцов. По Вашему желанию Вы можете указать только количество столбцов (параметр -columns); метод radio_group вычислит необходимое значение количества строк автоматически.

  Для того, чтобы включить в результирующую таблицу заголовки строк и столбцов, Вам необходимо использовать параметры **-rowheader** и **-colheader**. Обоим этим параметрам можно присвоить в качестве значения указатель на массив заголовков. Заголовки несут на себе чисто декоративную функцию. Они не оказывают никакого влияния на поведение радиокнопок - это просто именованные элементы.

При обработке формы радиокнопка, выбранная пользователем, может быть получена при помощи следующей конструкции:

```
$which_radio_button = $query->param('group_name');
```

Значение, возвращаемое методом `radio_group()` - это на самом деле массив элементов - кнопок. Вы можете затем присвоить его массиву и затем использовать его в списках, таблицах и в других созидательных целях:

```
@h = $query->radio_group(-name=>'group_name',-values=>\@values);
&use_in_creative_way(@h);
```

Примечание для использующих JAVASCRIPT: Метод `radio_group()` распознает параметр **-onClick**. Смотрите описание метода `checkbox_group()` для получения более детальной информации.

СОЗДАНИЕ КНОПКИ SUBMIT

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

Метод `submit()` создает кнопку, которая отправляет сценарию данные формы. Каждая форма должна иметь, по крайней мере, одну кнопку SUBMIT.

Параметры:

- Первый аргумент (-name) - необязательный. Вы можете присвоить кнопке имя, если Вы определили несколько различных кнопок в форме и хотите их различать. Имя также будет использовано в качестве надписи на кнопке. Будьте осторожны - некоторые старые браузеры не обрабатывают корректно метки кнопок и **никогда** не возвращают метку, присвоенную кнопке.
- Второй аргумент (-value) также необязателен. Он присваивает кнопке значение, которое будет передано Вашему сценарию в строке запроса.

Вы можете определить, какая именно кнопка была нажата, путем указания имени кнопки при вызове метода param():

```
$which_one = $query->param('button_name');
```

СОЗДАНИЕ КНОПКИ RESET (СБРОС)

```
print $query->reset
```

Метод `reset()` создает кнопку "reset''. Обратите внимание, что эта кнопка восстанавливает содержимое полей формы в состояние, в которое они были установлены при предыдущем вызове сценария, НЕОБЯЗАТЕЛЬНО к значениям, установленным по умолчанию.

Обратите внимание, что этот метод конфликтует со встроенной функцией Perl `reset()`. Используйте CORE::reset() для доступа к исходной функции Perl.

СОЗДАНИЕ КНОПКИ УСТАНОВКИ ПАРАМЕТРОВ ПО УМОЛЧАНИЮ

```
print $query->defaults('button_label')
```

Метод `defaults()` создает кнопку, нажатие на которую вызывает полный сброс параметров формы в значения установленные по умолчанию, отменяя любые изменения, когда-либо сделанные пользователем.

СОЗДАНИЕ СКРЫТОГО (HIDDEN) ПОЛЯ

```
print $query->hidden(-name=>'hidden_name',
                     -default=>['значение1','значение2'...]);
      -или-
print $query->hidden('hidden_name','значение1','значение2'...);
```

Метод `hidden()` создает поле формы , невидимое для пользователя (hidden). Это бывает полезно при передаче переменных, содержащих информацию о текущем состоянии от одного вызова сценария к следующему.

Параметры:

- Первый **обязательный** аргумент - имя поля (-name).
- Второй **обязательный** аргумент - значение поля по умолчанию (-default). При вызове с помощью именованных параметров Вы можете подставить в качестве значения скалярное значение Илии ссылку на список

Значение скрытого поля можно получить таким способом:

```
$hidden_value = $query->param('hidden_name');
```

Заметьте, что подобно всем другим формообразующим элементам, значение скрытого поля "обладает памятью". Если Вы хотите изменить значения, присвоенные скрытому полю уже после вызова сценария, Вы должны это сделать вручную, вызвав метод `param()`.

```
$query->param('hidden_name','подставляем','новые','значения');
```

СОЗДАНИЕ ГРАФИЧЕСКОЙ КНОПКИ

```
print $query->image_button(-name=>'button_name',
                           -src=>'/source/URL',
                           -align=>'MIDDLE');
  -или-
print $query->image_button('button_name','/source/URL','MIDDLE');
```

Метод `image_button()` генерирует код для вывода изображения, на котором можно щелкнуть, как на обычной кнопке. При щелчке на изображении метод возвращает сценарию координаты курсора в виде "button_name.x" и

"button_name.y", где "button_name" - это имя, которое Вы назначили этому графическому элементу

Для программистов, использующих JAVASCRIPT: Метод `image_button()` распознает параметр **-onClick**. Смотрите описание метода `checkbox_group()` для получения дополнительной информации.

Параметры:

- Первый аргумент (-name) - обязательный и определяет имя поля.
- Второй аргумент (-src) - также обязателен и определяет URL загружаемого изображения
- Третий необязательный элемент (-align) - тип выравнивания и может принимать значения TOP, BOTTOM или MIDDLE

Координаты курсора в момент щелчка могут быть получены следующим способом:

```
$x = $query->param('button_name.x');
$y = $query->param('button_name.y');
```

СОЗДАНИЕ КНОПКИ, КОТОРОЙ НАЗНАЧЕНА ПРОЦЕДУРА JAVASCRIPT

```
print $query->button(-name=>'button_name',
                     -value=>'Надпись на кнопке',
                     -onClick=>"do_something()");
  -или-
print $query->button('button_name',"do_something()");
```

Метод `button()` создает кнопку, которая совместима с JavaScript 1.0 (браузеры Netscape 2.0+ и MSIE 4.0+). При нажатии на эту кнопку будет выполнен фрагмент кода JavaScript, на который указывает параметр **-onClick**. В окне древнего браузера (к примеру, MSIE 3) этот элемент, возможно, даже не будет отображен.

ФАЙЛЫ COOKIES

Браузеры Netscape начиная с версии 1.1. и старше, а также все версии Internet Explorer, поддерживают так называемые "cookie", которые были разработаны для того, чтобы помочь сохранить текущее состояние в течение сессии браузера. Модуль CGI.pm содержит несколько методов для поддержки файлов cookie.

Cookie представляет собой пару имя=значение во многом аналогичную именованным параметрам в строке CGI запроса. Сценарии CGI создают один или несколько cookies и посылают их браузеру в заголовке HTTP. Браузер поддерживает список cookie, которые относятся к конкретному Web серверу, и возвращают его сценарию CGI в течение последующих взаимодействий.

В дополнение к обязательной паре имя=значение, каждый файл cookie может иметь несколько необязательных атрибутов:

1. **срок хранения**
   Это строка даты/времени (в специальном формате GMT) которая определяет конечный срок хранения файла у клиента. Файл cookie будет сохранен и отправлен Вашему сценарию вплоть до истечения срока хранения во всех последующих сессиях браузера (после выхода и последующего открытия браузера). Если же срок хранения не указан, файл cookie останется активным до тех пор, пока пользователь не выйдет из браузера.
2. **домен**
   Полное или частичное имя домена для которого имеет силу данный файл cookie. Браузер вернет файл cookie любому хосту, который соответствует частичному доменному имени. Например, если Вы указали в качестве параметра доменное имя ".capricorn.com", то браузер возвратит файл cookie серверу Web, запущенному на одной из машин "www.capricorn.com", "www2.capricorn.com","feckless.capricorn.com", и т.д.. Доменные имена должны содержать, по крайней мере, две точки для предотвращения попыток возвращения файла cookie доменам верхнего уровня, таким как, к примеру, ".edu". Если домен не указан, браузер вернет cookie только серверу, хост которого является родительским для данного cookie.
3. **путь**
   Если Вы укажите атрибут пути для cookie, браузер проверит URL вашего сценария на предмет соответствия указанному пути, прежде чем возвратить cookie. К примеру, если Вы указываете путь "/cgi-bin", то файл cookie будет возвращен сценариям "/cgi-bin/tally.pl", "/cgi-bin/order.pl", и "/cgi-bin/customer_service/complain.pl", но не сценарию "/cgi-private/site_admin.pl". По умолчанию, если путь установлен в "/", то файл cookie должен быть отправлен любому сценарию CGI Вашего сайта.
4. **Флаг "secure"**
   Если установлен атрибут "secure", файл cookie будет отправлен Вашему сценарию только в случае, когда запрос CGI отправлен в защищенном канале, таком как SSL.

Интерфейсом к файлам cookie служит метод **cookie()**:

```
$cookie = $query->cookie(-name=>'sessionID',
                         -value=>'xyzzy',
                         -expires=>'+1h',
                         -path=>'/cgi-bin/database',
                         -domain=>'.capricorn.org',
                         -secure=>1);
print $query->header(-cookie=>$cookie);
```

Метод **cookie()** создает новый файл cookie. Параметры метода:

-name
: Обязательный параметр - Имя cookie. Может быть произвольной строкой. Хотя браузеры и ограничивают имя файла cookie только непробельными алфавитно-цифровыми символами, модуль CGI.pm устраняет это ограничение путем скрытой от пользователя обработки файлов cookies.

-value
: Значение cookie. Может быть скалярным значением, ссылкой на массив или даже ссылкой на ассоциативный массив. К примеру, Вы можете сохранить ассоциативный массив целиком в файле cookie таким образом:

```
$cookie=$query->cookie(-name=>'family information',
                       -value=>\%childrens_ages);
```

-path
: Необязательный параметр - частичный путь для сценария, для которого этот файл cookie может быть возвращен, как описано выше.

-domain
: Необязательный параметр - часть доменного имени, для которого имеет силу этот файл cookie, как было описано выше.

-expires
: Необязательная дата - срок хранения этого файла cookie. Формат поля такой же, как и описанный в разделе, посвященном методу **header()**:

```
"+1h"  в течение часа с текущего момента
```

-secure
: Если параметр установлен в истину, то данный файл cookie будет использован только в защищенной SSL сессии.

Файл cookie созданный методом `cookie()` должен быть внедрен в заголовок HTTP строкой, возвращаемой методом `header()`:

```
print $query->header(-cookie=>$my_cookie);
```

Для создания нескольких файлов cookies, передайте `header()` ссылку на массив:

```
$cookie1 = $query->cookie(-name=>'riddle_name',
                          -value=>"Вопрос Сфинкса");
$cookie2 = $query->cookie(-name=>'answers',
                          -value=>\%answers);
print $query->header(-cookie=>[$cookie1,$cookie2]);
```

Для получения cookie, затребуйте его по имени путем вызова метода `cookie()` без параметра **-value**:

```
use CGI;
$query = new CGI;
%answers = $query->cookie(-name=>'answers');
# $query->cookie('answers') также будет работать!
```

Пространства имен cookie и CGI разделены. Если у вас есть параметр с именем 'answers' и cookie с именем 'answers', значения, полученные функциями `param()` и `cookie()` не зависят друг от друга. Однако нет ничего проще, чем превратить параметр CGI в cookie, и наоборот:

```
# превратим параметр CGI в cookie
$c=$q->cookie(-name=>'answers',-value=>[$q->param('answers')]);
# и наоборот
$q->param(-name=>'answers',-value=>[$q->cookie('answers')]);
```

Смотрите в качестве примера **cookie.cgi** для демонстрации эффективного использования файлов cookie.

```
#!/usr/bin/perl
# coockie.cgi
use CGI qw(:standard);
@ANIMALS=sort qw/лев тигр медведь кабан дикобраз хорек зебра гну страус
  эму корова козел ласка як цыпленок овца гиена дронт ящерица
  белка крыса мышь еж енот бабуин кенгуру бегемот жираф/;
# Восстановим предыдущее состояние из cookie.
  # Файл cookie был отформатирован в качестве ассоциативного массива
  # Каждому животному соответствует его количество в зоопарке.
  %zoo = cookie('animals');
# Recover the new animal(s) from the parameter 'new_animal'
  @new = param('new_animals');
# Если выбрано действие 'Добавить', то добавим новых животных в зоопарк
  # В противном случае удалим их
  foreach (@new) {
  if (param('action') eq 'Добавить') {
  $zoo{$_}++;
  } elsif (param('action') eq 'Удалить') {
  $zoo{$_}-- if $zoo{$_};
  delete $zoo{$_} unless $zoo{$_};
  }
  }
# Добавить новых зверей к старым, затем поместитть их в cookie
  $the_cookie = cookie(-name=>'animals',
  -value=>\%zoo,
  -expires=>'+1h');
# Напечатать заголовок, содержащий cookie и срок хранения..
  print header(-cookie=>$the_cookie);
# Сейчас мы готовы к созданию нашей страницы HTML.
  print start_html('Взломщик зоопарка');
print <<EOF;
  <h1>Взломщик зоопарка</h1>
  Выберите животное, которое Вы хотите поместить в зоопарк, и нажмите кнопку
  "Добавить". Возвратитесь на страницу в любое время в течение часа и
  список зверей в зоопарке возродится. Вы даже можете полностью выйти из браузера!
<p>
  Попытайтесь добавить в список одно и то же животное несколько раз
  Не напоминает ли это Вам некую тележку для покупок (shopping cart)?
  <p>
  <em>Данный сценарий работает только с браузерами Netscape (начиная с версиит 1.1
  и MS Internet Explorer версии 4 и старше</em>
```

```
   <p>
   <center>
   <table border=1>
   <tr><th>Добавить/Удалить<th>Имеется на текущий момент
EOF
   ;
print "<tr><td>",start_form;
   print scrolling_list(-name=>'new_animals',
   -values=>[@ANIMALS],
   -multiple=>1,
   -override=>1,
   -size=>10),"<br>";
   print submit(-name=>'action',-value=>'Удалить'),
   submit(-name=>'action',-value=>'Добавить');
   print end_form;
print "<td>";
   if (%zoo) { # make a table
   print "<ul>\n";
   foreach (sort keys %zoo) {
   print "<li>$zoo{$_} $_\n";
   }
   print "</ul>\n";
   } else {
   print "<strong>Зоопарк пуст.</strong>\n";
   }
   print "</table></center>";
print <<EOF;
   <hr>
   <ADDRESS>Lincoln D. Stein</ADDRESS><BR>
EOF
   ;
   print end_html;
```

РАБОТА С ФРЕЙМАМИ

Сценарии на основе модуля CGI.pm могут писать поток в различные панели и окна браузера, используя механизм фреймов HTML 4. Имеется три способа программного определения новых фреймов:

1. **Создание <Frameset> - документа**
   После генерации заголовка HTTP, вместо создания стандартного документа HTML при помощи вызова `start_html()`, создается FRAMESET - документ, который определяет фреймы на странице. Определите Ваши методы `script` (с соответствующими параметрами) в качестве операнда SRC для каждого из фреймов.

   В модуле CGI.pm не имеется специфической поддержки для создания секций <FRAMESET>, однако генерируемый текст HTML очень прост. Смотрите документацию по фреймам на сервере Netscape для получения дополнительной информации.

   http://home.netscape.com/assist/net_sites/frames.html

2. **Определите окно вывода документа в заголовке HTTP**
   Вы можете указать параметр **-target** в методе `header()`:

   ```
   print $q->header(-target=>'ResultsWindow');
   ```

   Это предложение предпишет браузеру загрузить вывод Вашего сценария во фрейм с именем ResultsWindow. Если фрейм с таким именем не существует, браузер откроет новое окно и загрузит в него результат выполнения сценария. Имеется целый набор "волшебных" имен, которые Вы можете использовать в качестве результирующих имен окон. Смотрите документацию по фреймам на сервере Netscape для получения дополнительной информации.

3. **Определите окно вывода документа в тэге <FORM>**
   Вы можете определить фрейм, в который загружается сама форма (описываемая тэгами FORM). С помощью CGI.pm Вы можете написать примерно такое предложение:

   ```
   print $q->startform(-target=>'ResultsWindow');
   ```

   Когда Ваш сценарий вызывается формой, его вывод направляется во фрейм с именем "ResultsWindow". Если фрейм с таким именем не существует, создается новое окно.

Сценарий frameset.cgi, приведенный ниже, показывает один из способов создания страниц, в котором заполняемая форма и результат сосуществуют в расположенных рядом фреймах.

```
#!/usr/local/bin/perl

use CGI;
$query = new CGI;
print $query->header;
$TITLE="Пример работы с фреймами";

# Мы используем информацию о пути, по которому расположен сценарий,
# для того чтобы :
# (1) создать фреймсет
# (2) создать форму запроса
# (3) создать результат ответа на запрос

$path_info = $query->path_info;

# Если не задана информация о пути, создаем
# два расположенных рядом фрейма фреймсета
if (!$path_info) {
   &print_frameset;
   exit 0;
}

# В этом месте мы создаем либо ноую форму запроса
# либо ответное действие.
&print_html_header;
&print_query if $path_info=~/query/;
&print_response if $path_info=~/response/;
&print_end;


# Создать фреймсет
sub print_frameset {
   $script_name = $query->script_name;
   print  <<EOF;
<html><head></head>
<frameset cols="50,50">
<frame src="$script_name/query" name="query">
<frame src="$script_name/response" name="response">
</frameset>
EOF
   ;
   exit 0;
}

sub print_html_header {
   print $query->start_html($TITLE);
}

sub print_end {
   print qq{<P><hr>};
   print $query->end_html;
}

sub print_query {
   $script_name = $query->script_name;
   print "<H1>Использование фреймсета - страницы, использующей фреймы</H1>\n";
   print $query->startform(-action=>"$script_name/response",-TARGET=>"response");
   print "Ваше имя? ",$query->textfield('name');
   print "<P>Любимое слово?<P>",
   $query->checkbox_group(-name=>'words',
                          -values=>['Хрю','Гав','Мяу','Муу']);
```

```
    print "<P>Ваш любимый цвет? ",
    $query->popup_menu(-name=>'color',
                       -values=>['красный','зеленый','синий','янтарный']),
    "<P>";
    print $query->submit;
    print $query->endform;
}

sub print_response {
    print "<H1>Результат создания фреймсета</H1>\n";
    unless ($query->param) {
      print "<b>Запрос еще не выполнен.</b>";
      return;
    }
    print "Ваше имя <EM>",$query->param(name),"</EM>\n";
    print "<P>Любимые слова: <EM>",join(", ",$query->param(words)),"</EM>\n";
    print "<P>Ваш любимый цвет<EM>",$query->param(color),"</EM>\n";
}
```

ОГРАНИЧЕННАЯ ПОДДЕРЖКА СТИЛЕВОГО ОФОРМЛЕНИЯ (CSS)

Модуль CGI.pm имеет ограниченную поддержку стилевого оформления (CSS), описанную в стандарте HTML3. Для внедрения таблицы стилевого оформления в создаваемый Вами документ HTML, вызовите метод `start_html()` с параметром **-style**. Значением этого параметра может быть скаляр, в таком случае таблица стилевого оформления внедряется непосредственно в секцию <STYLE> документа HTML или может быть ссылкой на хэш. Во втором случае Вам потребуется определить хэш с одним или более параметров **-src** или **-code**. **-src** указывает на URL, по которому можно найти внешний файл описания стилевого оформления. **-code** указывает на скаляр, внедряемый в секцию <STYLE> документа HTML. Определения стилей, сделанные в параметре **-code**, заменяют стили с такими же именами, определенные в параметре **-src**, отсюда и происходит название "каскадные".

Вы также можете определить тип таблицы стилевого оформления путем добавления дополнительного необязательного параметра **-type** к хэшу, на который указывает **-style**. Если он не указан, то по умолчанию подразумевается значение 'text/css'.

Для того чтобы сослаться на стиль оформления внутри тела документа, добавьте параметр **-class** к любому элементу HTML:

```
print h1({-class=>'Fancy'},'Welcome to the Party');
```

или определить стили "на лету" при помощи параметра **-style**:

```
print h1({-style=>'Color: red;'},'Welcome to Hell');
```

Вы также можете использовать новый элемент **span()** для того чтобы применить стиль к секции текста:

```
print span({-style=>'Color: red;'},
        h1('Добро пожаловать в АД!'),
        "Where did that handbasket get to?"
        );
```

Обратите внимание на то, что Вы обязаны импортировать набор деклараций ":html3"' в директиве use, чтобы получить доступ к методу **span()**. Ниже приведен краткий (очень неряшливо написанный) пример использования CSS. Полная спецификация CSS доступна по адресу http://www.w3.org/pub/WWW/TR/Wd-css-1.html, где Вы можете получить более подробную информацию об использовании стилевого оформления документов HTML.

```
use CGI qw/:standard :html3/;
# в данном примере таблица стилевого оформления внедрена в текст страницы
$newStyle=<<END;
```

```
<!--
P.Tip {
    margin-right: 50pt;
    margin-left: 50pt;
    color: red;
}
P.Alert {
    font-size: 30pt;
    font-family: sans-serif;
  color: red;
}
-->
END
print header();
print start_html( -title=>'CGI.pm со стилевым оформлением',
                -style=>{-src=>'http://www.capricorn.com/style/st1.css',
                        -code=>$newStyle}
                );
print h1('CGI.pm со стилевым оформлением'),
      p({-class=>'Tip'},
        "Прежде чем играть с этим примероа, прочтите-ка документацию
                              о таблицах стилевого оформления!"),
      span({-style=>'color: magenta'},
          "Но-но! Без рук!",
          p(),
          "Yay!"
          );
print end_html;
```

ОТЛАДКА

Если Вы запускаете сценарий из командной строки или в отладчике perl , Вы можете передать сценарию список ключевых слов (параметров) или пар ПАРАМЕТР=ЗНАЧЕНИЕ в командной строке или со стандартного ввода (Вам не нужно беспокоиться о том как сценарий получит переменные окружения). Вы можете передать ключевые слова следующим образом:

```
your_script.pl слово1 слово2 слово3
```

или так:

```
your_script.pl keyword1+keyword2+keyword3
```

или:

```
your_script.pl name1=value1 name2=value2
```

или так:

```
your_script.pl name1=value1&name2=value2
```

или даже в качестве строк стандартного ввода, разделенных символами новой строки.

При отладке Вы можете использовать кавычки и обратный слэш для ввода esc-последовательностей способом подобным используемому интерпретатору shell, что позволит Вам вставить пробелы и прочие подобные забавные символы в пары ПАРАМЕТР = ЗНАЧЕНИЕ:

```
your_script.pl "name1='I am a long value'" "name2=two\ words"
```

ФОРМАТИРОВАННЫЙ ВЫВОД (DUMPING) ВСЕХ ПАР ИМЯ/ЗНАЧЕНИЕ

Метод `dump()` выдает в качестве результата строку, состоящую из всех пар имя/значение для данного запроса, причем красиво отформатированного в качестве вложенного списка. Это очень полезно для целей отладки:

```
print $query->dump
```

Выдает в результате нечто подобное:

```
<UL>
<LI>имя1
   <UL>
   <LI>значение1
   <LI>значение2
   </UL>
```

```
<LI>имя2
   <UL>
   <LI>значение1
   </UL>
</UL>
```

В качестве сокращения, Вы можете включить объект CGI целиком в строку и она будет заменена красиво отформатированным текстом HTML, как было показано выше:

```
$query=new CGI;
print "<H2>Текущие значения</H2> $query\n";
```

ИЗВЛЕЧЕНИЕ ПЕРЕМЕНЫХ ОКРУЖЕНИЯ

Некоторые из наиболее полезных переменных окружения могут быть получены при помощи этого интерфейса. Методы описаны ниже:

Accept()

Возвращает список типов MIME, которые может принять браузер-клиент. Если Вы передаете этому методу единичный аргумент, соответствующий типу MIME, как в следующем случае: $query->Accept('text/html'), он возвратит в качестве значения число с плавающей точкой в соответствии с предпочтениями браузера для этого типа данных - от 0.0 (не принимается) до 1.0. Типы, описанные с помощью регулярных выражений - globs (например, text/*) в списке воспринимаемых типов браузера обрабатываются корректно.

Заметьте, что наименование метода стало писаться с большой буквы при переходе с версии 2.43 на 2.44, во избежание конфликта со встроенной функцией Perl `accept()`.

raw_cookie()

Возвращает переменную HTTP_COOKIE, расширение HTTP примененное в браузерах Netscape начиная с версии 1.1 и старше, а также во всех версиях Internet Explorer. Файлы cookie имеют специальный формат, а вызов этого метода просто возвращает содержимое cookie в необработанной форме (т.н. сырое тесто для печенья :-0) (?cookie dough). Смотрите описание метода `cookie()` о методах записи и получения подготовленных "печенюшек" - coockies.

Вызванный без параметров, метод `raw_cookie()` возвращает упакованную структуру cookie. Вы можете разделить ее на отдельные cookie при помощи расщепления строки по символу "точка с запятой" (;)'. Если метод вызван с именем cookie в качестве параметра, получает требуемый cookie, в форме, очищенной от esc-последовательностей (**unescaped**). Для получения имен Вы можете воспользоваться обычным методом `cookie()`, или использовать метод `raw_fetch()` модуля CGI::Cookie.

user_agent()

Возвращает переменную HTTP_USER_AGENT. Если Вы передаете этому методу один аргумент, он попытается найти в этой переменной соответствие шаблону, позволяя Вам делать нечто вроде $query->user_agent(netscape);

path_info()

Возвращает дополнительную информацию о пути из URL сценария. Например, вызов сценария $query->`path_info()`, расположенного по адресу /cgi-bin/your_script/additional/stuff возвратит в качестве результата "additional/stuff".

ПРИМЕЧАНИЕ: В сервере Microsoft Internet Information Server имеется ошибка, связанная с обработкой дополнительной информации о пути. Если Вы используете библиотеку Perl DLL, сервер IIS попытается выполнить дополнительную строку, содержащую путь, в качестве сценария Perl. Если Вы используете обычное соответствие типов файлов, то информация о пути будет присутствовать в переменной окружения , но будет неверной. Лучшим выходом из этой ситуации будет отказ от использования любой дополнительной информации о пути в сценариях CGI, предназначенным для использования с IIS.

path_translated()

Действует аналогично `path_info()`,но возвращает дополнительную информацию о пути, по которому находится сценарий в виде абсолютного адреса, например,"/usr/local/etc/httpd/htdocs/additional/stuff".

Сервер Microsoft IIS неправильно транслирует дополнительный путь и в этом случае также.

remote_host()

Возвращает имя удаленного хоста, или адрес IP. если имя хоста невозможно определить.

script_name()

Возвращает имя сценария в виде частичного URL, для использования во внутренних ссылках

referer()

Возвращает URL страницы, которая была открыта в строке браузера перед вызовом Вашего сценария. Доступно не для всех браузеров.

auth_type ()

Возвращает метод авторизации/верификации используемый данным сценарием, если таковой имеется.

server_name ()

Возвращает имя сервера, как правило, это имя машины-хоста.

virtual_host ()

При использовании виртуальных серверов, возвращает имя хоста (сервера), с которым пытается соединиться браузер.

server_software ()

Возвращает имя и номер версии программы - сервера.

remote_user ()

Возвращает авторизацию/верифицированное имя, используемое для идентификации пользователя, если сам сценарий защищен.

user_name ()

Пытается определить имя удаленного пользователя при помощи множества различных методик. Эта возможность действует только для старых браузеров, таких как Mosaic. Новые браузеры не сообщают сценарию имя пользователя по соображениям конфиденциальности!

request_method()

Возвращает метод, используемый для вызова вашего сценария, обычно один из 'POST', 'GET' или 'HEAD'.

content_type()

Возвращает тип содержимого данных (content_type), переданных сценарию методом POST, обычно имеет значение "multipart/form-data" или "application/x-www-form-urlencoded"

http()

Вызванный без аргументов, возвращает список переменных окружения HTTP, включая HTTP_USER_AGENT, HTTP_ACCEPT_LANGUAGE и HTTP_ACCEPT_CHARSET, которые соответствуют полям заголовка HTTP-запроса, имеющим такие же имена. Вызванный с аргументом - именем поля заголовка HTTP, возвращает значение этого поля. Регистр символов и использование тире вместо знаков подчеркивания не существенно.

Например, следующие три строки эквивалентны:

```
$requested_language = $q->http('Accept-language');
$requested_language = $q->http('Accept_language');
$requested_language = $q->http('HTTP_ACCEPT_LANGUAGE');
```

https()

То же самое, что и *http()*, но функционирует в пространстве переменных окружения HTTPS, которое возникает в случае, когда установлен протокол SSL. Может быть использован для определения того, включен или нет режим SSL.

ИСПОЛЬЗОВАНИЕ NPH СЦЕНАРИЕВ (СЦЕНАРИЕВ С НЕАНАЛИЗИРУЕМЫМИ ЗАГОЛОВКАМИ)

NPH, или "no-parsed-header'', сценарии (сценарии с неанализируемыми заголовками) полностью обходят сервер, отправляя заголовок HTTP непосредственно браузеру. Эта технология дает небольшой выигрыш в производительности, но в основном используется для обработки расширений HTTP, напрямую не поддерживаемых сервером, таких как выталкивание страниц (server push) и заголовки PICS

Серверы используют разнообразные соглашения для распознавания сценариев CGI как NPH - сценариев. Многие Unix-серверы ищут в начале названия сценария префикс "nph-". Сервер WebSTAR для Macintosh и Microsoft's Internet Information Server, напротив, пытаются решить, является ли программа сценарием NPH, анализирую первую строку вывода сценария.

Модуль CGI.pm поддерживает сценарии NPH, работая в специальном NPH режиме. Работая в этом режиме , CGI.pm выдаст дополнительную информацию в заголовок HTTP при вызове методов `header()` и `redirect()`.

Microsoft Internet Information Server требует от сценария, чтобы он выполнялся в режиме NPH. Начиная с версии 2.30, CGI.pm автоматически определит, что сценарий выполняется в среде IIS и переведет себя в режим NPH. Вам не нужно делать это вручную, хотя и не нанесет вреда, если Вы это сделаете.

Имеется ряд способов перевести модуль CGI.pm в режим NPH:

Директивой use

Просто добавьте прагму "-nph" к списку символов, импортируемых в Ваш сценарий:

```
use CGI qw(:standard -nph)
```

При помощи вызова метода nph():

Вызовите метод **nph()** с ненулевым параметром в любой точке сценария после директивы "use CGI.pm";.

```
CGI->nph(1)
```

Указанием параметра -nph в методах header() и redirect():

```
print $q->header(-nph=>1);
```

Выталкивание страниц (Server Push)

Модуль CGI.pm обеспечивает три простых функции для создания многосекционных документов в виде, необходимом для реализации метода выталкивания страниц. Эти функции любезно предоставлены Эдом Джорданом <ed@fidalgo.net>. Для импортирования этих функций в пространство имен текущего сценария, укажите в качестве импортируемого набора "push" в директиве use CGI. Мы также советуем Вам установить для сценария режим NPH,

а также присвоить переменной $| значение 1 во избежание проблем с буферизацией.

Ниже приведен простой сценарий, который демонстрирует технологию выталкивания страниц (push):

```
#!/usr/local/bin/perl
use CGI qw/:push -nph/;
$| = 1;
print multipart_init(-boundary=>'----------а вот и мы!');
while (1) {
   print multipart_start(-type=>'text/plain'),
        "Точное время",scalar(localtime),"\n",
        multipart_end;
   sleep 1;
}
```

Данный сценарий инициализирует механизм серверного выталкивания страниц при помощи вызова **multipart_init()**.Затем сценарий входит в бесконечный цикл, в котором он начинает новую секцию многосекционного документа при помощи вызова **multipart_start()**, печатает текущее локальное время и закрывает секцию, вызывая **multipart_end()**. Затем он засыпает на секунду и продолжает выполнение с начала цикла.

multipart_init()

```
multipart_init(-boundary=>$boundary);
```

Инициализирует многосекционную структуру. Аргумент *-boundary* определяет, какая строка - разделитель MIME будет использована для разделения секций документа. Если аргумент не указан, модуль CGI.pm выбирает значимое значение для разделителя самостоятельно.

multipart_start()

```
multipart_start(-type=>$type)
```

Начинает новую секцию многосекционного документа, которая имеет указанный тип MIME. Если тип MIME не указан, по умолчанию принимается тип text/html

multipart_end()

```
multipart_end()
```

Завершает секцию. Вы должны помнить о необходимости вызова `multipart_end()` по одному разу для каждого открывающего метода `multipart_start()`.

Пользователи, заинтересованные в серверных приложениях, выталкивающих страницы, должны, конечно, обратить свое внимание на модуль CGI::Push.

Как избежать хакерских атак

Потенциальная проблема, связанная с CGI.pm, состоит в том, что, по умолчанию, модуль пытается обработать данные, передаваемые форме методом POST, вне зависимости от того, насколько велик их объем. Коварный хакер может атаковать Ваш сайт, передавая вашему CGI сценарию методом POST громадную порцию информацию размеров во много мегабайт. Модуль CGI.pm will попытается прочесть весь поток POST в переменную, которая вырастет до огромных размеров, пока не заполнит всю имеющуюся память. Во время попытки резервирования памяти сценарием производительность системы может существенно снизиться. Это служит формой противодействия хакерским атакам.

Другим возможным видом атаки может быть попытка удаленного пользователя заставить модуль CGI.pm загрузить громадный файл. CGI.pm воспримет выгружаемый файл и сохранит его в промежуточной директории, даже если Ваш сценарий не собирается принимать выгружаемый файл. Модуль CGI.pm автоматически удалит файл после завершения выгрузки, но в течение этого времени удаленный пользователь может быстро заполнить дисковое пространство сервера, что приведет к возникновению проблем в работе других программ.

Лучшим способом избежания хакерских атак является ограничение количества доступной памяти, использования времени процессора и дискового пространства, которое может использовать сценарий CGI. Некоторые серверы Web поставляются со встроенными возможностями выполнения такой задачи. В других случаях Вы можете использовать команды shell - *limit* или *ulimit* для установки предельных значений ресурсов, которые может использовать CGI.

Модуль CGI.pm также имеет простую встроенную защиту против хакерских атак, но Вы обязаны активировать ее перед использованием. Она представляет собой две глобальных переменных в пространстве имен CGI:

$CGI::POST_MAX

Если этой переменной присвоено ненулевое значение, модуль CGI.pm устанавливает предельный размер в байтах размеров информационного потока, передаваемого формой при помощи метода POST. Если CGI.pm обнаруживает, что поток данных, передаваемый методом POST больше установленного предела, он немедленно прекращает выполнение сценария с выдачей сообщения об ошибке. Это значение влияет как на обычный поток POST, так и на передачу многосекционных документов методом POST, ограничивая максимальный размер выгружаемой информации. Вам нужно присвоить этой переменной достаточно большое разумное значение, к примеру 1 мегабайт.

$CGI::DISABLE_UPLOADS

Если переменной присвоено ненулевое значение, то выгрузки файлов полностью запрещены. Другие значения полей заполненной формы будут вести себя как обычно.

Вы можете использовать эти переменные двумя способами.

1. **1. Для каждого сценария отдельно**
   Вставьте операторы присваивания данным переменным в самое начало сценария, сразу после операторов "use":

```
use CGI qw/:standard/;
use CGI::Carp 'fatalsToBrowser';
$CGI::POST_MAX=1024 * 100;  # не более 100K может быть передано методом POST
$CGI::DISABLE_UPLOADS = 1;  # запретить выгрузку файлов
```

2. **2. Глобально для всех сценариев**
   Откройте CGI.pm, найдите место, где определяются переменные $POST_MAX и $DISABLE_UPLOADS, и присвойте им желаемые значения. Вы найдете их ближе к началу файла в подпрограмме initialize_globals().

Попытка передать методом POST объем данных более чем $POST_MAX байт приведет к тому, что *param()* возвратит пустой список параметров CGI. Вы можете определить наступление такого события, проверив его при помощи вызова метода *cgi_error()*, либо после создания объекта CGI либо, если Вы используете функционально ориентированный интерфейс, при помощи первоначального вызова <param()>. Если POST был перехвачен по причине превышения максимального объема передаваемых данных, то `cgi_error()` возвратит сообщение "413 POST too large''.

Это сообщение об ошибке фактически определено протоколом HTTP, и специально разработано для того, чтобы возвратить его браузеру в качестве кода состояния сценария CGI. Например:

```
$uploaded_file = param('upload');
if (!$uploaded_file && cgi_error()) {
   print header(-status=>cgi_error());
   exit 0;
}
```

Однако, не вполне ясно, знает ли тот или иной браузер, что делать с этим кодом состояния. Более приемлемым способом представляется создание страницы HTML, которая предупреждает пользователя о возникшей проблеме.

КНИГА О CGI.pm

Книга "The Official Guide to CGI.pm", написанная Линкольном Штейном напичкана всяческими методиками и трюками, связанными с использованием данного модуля, а также с информацией о внутренней структуре модуля, которую вряд ли можно найти где-нибудь еще. Она может быть найдена на книжных полках в американских библиотеках. Ее можно приобрести и в Amazon.com Посетите Web-сервер, сопровождающий эту книгу:
http://www.wiley.com/compbooks/stein/

Если у кого-то имеется эта книга в оригинале, готов перевести (прим. переводчика)

CGI.pm и проблема 2000 года

Версии CGI.pm до 2.36 были подвержены проблеме 2000 года в обработке файлов cookie. Год в сроке хранения файлов cookie выражался двумя цифрами, как было описано в протоколе Netscape, действовавшем в тот момент. Сейчас протокол cookie уже исправлен и существует уверенность, что модуль CGI.pm, начиная с версии 2.36 и старше, избавлен от проблемы 2000 года.

СОВМЕСТИМОСТЬ С CGI-LIB.PL

Для облегчения переноса существующих программ, которые используют cgi-lib.pl предоставлена подпрограмма ReadParse. Процесс переноса очень прост:

СТАРАЯ ВЕРСИЯ

```
require "cgi-lib.pl'';
&ReadParse;
print "The value of the antique is $in{antique}.\n'';
```

НОВАЯ ВЕРСИЯ

```
use CGI;
CGI::ReadParse;
print "The value of the antique is $in{antique}.\n'';
```

Процедура CGI.pm`ReadParse()` создает а связанную переменную с именем %in, при помощи которой можно получить значения переменных запроса. Подобно ReadParse, Вы также можете использовать свою собственную переменную. Редко используемые возможности ReadParse, такие, как создание переменных @in и $in, не поддерживаются.

После первого использования ReadParse, Вы можете сам получить объект запроса следующим образом:

```
$q = $in{CGI};
print $q->textfield(-name=>'wow',
                    -value=>'это в самом деле работает?');
```

Это позволяет Вам начать пользоваться более продвинутыми возможностями, заложенными в CGI.pm не переписывая свои старые сценарии с самого начала.

ИНФОРМАЦИЯ ОБ АВТОРЕ И РАСПРОСТРАНЕНИЕ МОДУЛЯ

Текст модуля может быть свободно модифицирован, но автор требует, чтобы соглашение об авторских правах было приложено распространяемому к файлу Вы можете свободно модифицировать данный модуль, однако если Вы распространяете измененную версию, приложите к ней, пожалуйста примечание о сделанных изменениях.

Эта библиотека является бесплатным программным обеспечением; Вы можете распространять и/или модифицировать ее на тех же условиях, что и сам язык Perl.

СПИСОК РАССЫЛКИ CGI-perl

Список рассылки CGI Perl закрыт и вряд ли будет возрожден. Адресуйте, пожалуйста, свои вопросы в конференцию comp.infosystems.www.authoring.cgi если они связаны с протоколом CGI или использованием CGI.pm или в конференцию comp.lang.perl.misc - об использовании языка Perl. Прочтите тщательно настоящую документацию, прочтите FAQ (ответы на часто задаваемые вопросы) этих телеконференций и посмотрите предыдущую переписку, прежде чем написать письмо в конференцию. Участники конференции недружелюбно относятся к наглым ленивым новичкам, не придерживающимся сетевого этикета!

РАЗРАБОТЧИКИ

Выражаем огромную благодарность:

Matt Heffron (heffron@falstaff.css.beckman.com)
James Taylor (james.taylor@srs.gov)
Scott Anguish <sanguish@digifix.com>
Mike Jewell (mlj3u@virginia.edu)
Timothy Shimmin (tes@kbs.citri.edu.au)
Joergen Haegg (jh@axis.se)
Laurent Delfosse (delfosse@delfosse.com)
Richard Resnick (applepi1@aol.com)
Craig Bishop (csb@barwonwater.vic.gov.au)
Tony Curtis (tc@vcpc.univie.ac.at)
Tim Bunce (Tim.Bunce@ig.co.uk)
Tom Christiansen (tchrist@convex.com)
Andreas Koenig (k@franz.ww.TU-Berlin.DE)
Tim MacKenzie (Tim.MacKenzie@fulcrum.com.au)
Kevin B. Hendricks (kbhend@dogwood.tyler.wm.edu)
Stephen Dahmen (joyfire@inxpress.net)
Ed Jordan (ed@fidalgo.net)
David Alan Pisoni (david@cnation.com)
Doug MacEachern (dougm@opengroup.org)
Robin Houston (robin@oneworld.org)
...и еще многим-многим-многим...
за ценные предложения и локализацию ошибок.

ЗАКОНЧЕННЫЙ ПРИМЕР ПРОСТОГО СЦЕНАРИЯ ОБРАБОТКИ ФОРМЫ

```
#!/usr/local/bin/perl
use CGI;
$query = new CGI;
print $query->header;
print $query->start_html("Пример обработки формы с помощью CGI.pm ");
print "<H1> Простая форма созданная при помощи CGI.pm</H1>\n";
&print_prompt($query);
&do_work($query);
&print_tail;
print $query->end_html;
sub print_prompt {
   my($query) = @_;
   print $query->startform;
   print "<EM>Ваше имя?</EM><BR>";
   print $query->textfield('name');
   print $query->checkbox('Это не мое настоящее имя');
   print "<P><EM>Где можно увидеть английских воробьев?</EM><BR>";
   print $query->checkbox_group(
                    -name=>'Sparrow locations',
                    -values=>[Англия,Франция,Испания,Азия,Тьмутаракань],
                    -linebreak=>'yes',
                    -defaults=>[Англия,Азия]);
   print "<P><EM>Как далеко они могут улететь?</EM><BR>",
       $query->radio_group(
             -name=>'how far',
             -values=>['10 футов','1 миля','10 миль','в самом деле далеко'],
             -default=>'1 миля');
   print "<P><EM>Ваш любимый цвет?</EM>  ";
   print $query->popup_menu(-name=>'Color',
                      -values=>['черный','коричневый','красный',желтый'],
                      -default=>'red');
   print $query->hidden('Reference','Monty Python and the Holy Grail');
   print "<P><EM>What have you got there?</EM><BR>";
   print $query->scrolling_list(
              -name=>'possessions',
              -values=>['Кокосовый орех','Чаша Грааля','Икона',
                     'Меч','Билет'],
              -size=>5,
              -multiple=>'true');
   print "<P><EM>Any parting comments?</EM><BR>";
   print $query->textarea(-name=>'Comments',
                      -rows=>10,
                      -columns=>50);
   print "<P>",$query->reset;
   print $query->submit('Action','Крик');
   print $query->submit('Action','Вопль');
   print $query->endform;
   print "<HR>\n";
}
sub do_work {
   my($query) = @_;
   my(@values,$key);
   print "<H2>Текущие значения параметров этой формы</H2>";
   foreach $key ($query->param) {
      print "<STRONG>$key</STRONG> -> ";
```

```
    @values = $query->param($key);
    print join(", ",@values),"<BR>\n";
  }
}
sub print_tail {
   print <<END;
<HR>
<ADDRESS>Lincoln D. Stein</ADDRESS><BR>
 END
}
```

ОШИБКИ

Этот модуль в процессе роста стал очень большим и цельным. Более того, он выполняет огромное количество действий, к примеру, обработку URL, разбор входного потока CGI, генерацию HTML и т.д., которые также выполняются модулями LWP. Они должны быть отвергнуты в пользу модулей CGI::* , но автор почему-то продолжает над ними работать (автор один и тот же ;-).

Обратите внимание на то, что текст модуля тщательно "вылизан", чтобы избежать ложных предупреждений из программ, запущенных с ключом **-w**.

СООБЩЕНИЯ ОБ ОШИБКАХ

Отправляйте, пожалуйста сообщения о найденных ошибках и свои комментарии по адресу:lstein@cshl.org. При отправке укажите, пожалуйста, следующую информацию:

- версию CGI.pm (perl -MCGI -e 'print $CGI::VERSION')
- версию Perl (perl -v)
- наименование и версию web-сервера,
- наименование и версию используемой операционной системы, если возможно,
- наименование и версию используемого Вами web-браузера
- короткий тестовый сценарий, демонстрирующий проблему (не более 30 строк)

Очень важно, чтобы автор получил эту информацию, чтобы он мог помочь Вам.

СМОТРИТЕ ТАКЖЕ

Документацию по модулям: CGI::Carp, URI::URL, CGI::Request, CGI::MiniSvr, CGI::Base, CGI::Form, CGI::Push, CGI::Fast, CGI::Pretty

CGI - модуль, реализующий функции Common Gateway Interface ОПИСАНИЕ ВЕРСИИ 2.56