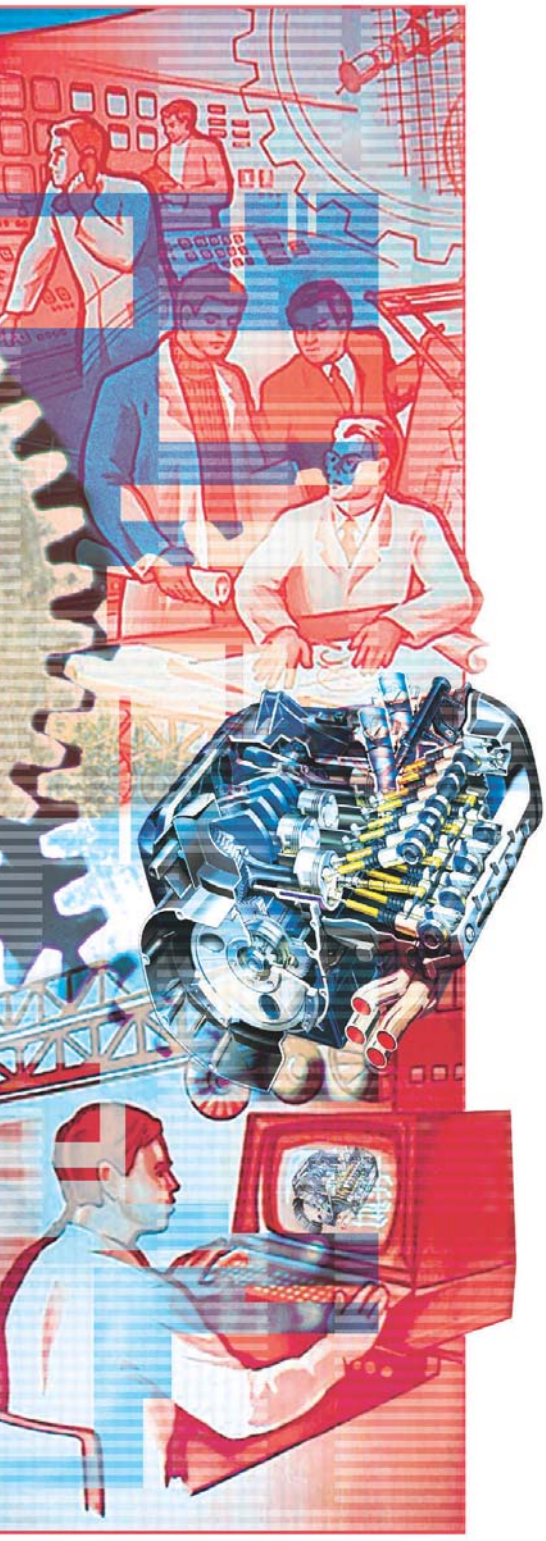
Работа со скриптами

Нужно ли форсировать движок?



Стремительное развитие web-технологий зачастую приводит к тому, что разработчики начинают применять достаточно сложные решения при создании своих ресурсов. Разумеется, во многих случаях такой подход оправдан. Но давайте подумаем, всегда ли? Подчас движок сайта можно сделать, используя достаточно простые классические средства и методы.

Под движком (engine) обычно подразумевают некую «обвязку» сайта, позволяющую облегчить работу web-мастера по обновлению страниц, по смене их оформления, в той или иной мере автоматизировать рутинную работу.

Как правило, движок выполняется в виде набора серверных приложений (чаще всего CGI-скриптов), что позволяет динамически, на лету, генерировать страницы, содержимое которых в данный момент времени может зависеть от многих факторов.

Степень сложности движка зависит от назначения и задач самого сайта. Иногда бывает достаточно использовать парутройку скриптов размером по килобайту каждый, но движки солидных web-проектов с многотысячной суточной посещаемостью порой представляют собой весьма и весьма дорогостоящее коммерческое программное обеспечение, базирующееся на возможностях серьезных СУБД архитектуры клиент-сервер, таких как Oracle или Informix. Как говорится в таких случаях, большому кораблю — большое плаванье...

Тем не менее при разработке небольших web-ресурсов со скромным бюджетом или отсутствием такового создание движка на основе приложений, исполняемых на сервере, может быть нежелательным. Причин тому как минимум три.

- Во-первых, недорогие тарифные планы коммерческих провайдеров (а тем более бесплатный хостинг) зачастую не предоставляют возможности исполнять CGI-скрипты, а порой даже не разрешают использовать элементарные SSI.
- Во-вторых, использование CGI и SSI существенно нагружает сервер, особенно если машина не слишком мощная, что тут же не лучшим образом сказывается на времени отклика и продолжительности загрузки страниц.
- В-третьих, серверные приложения зачастую таят в себе угрозы безопасности — в соответствующей литературе описаны десятки способов атак на web-сервер посредством размещенных на нем CGI-скриптов. Причем и на старуху бывает прорuха — даже грамотно написанные приложения (что, к слову, встречается крайне редко, чу- »

» ть ли не в единичных случаях) потенциально могут стать уязвимыми, ведь даже опытный программист — тоже человек, и он физически не может предусмотреть все и вся.

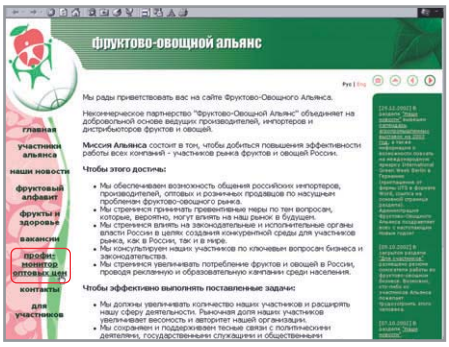
В пику прогрессу

Разработчики со стажем не напрасно (в основном в силу последних двух причин) рекомендуют воздерживаться от динамических страниц везде, где только это возможно, используя CGI и другие серверные технологии только там, где без них никак нельзя обойтись.

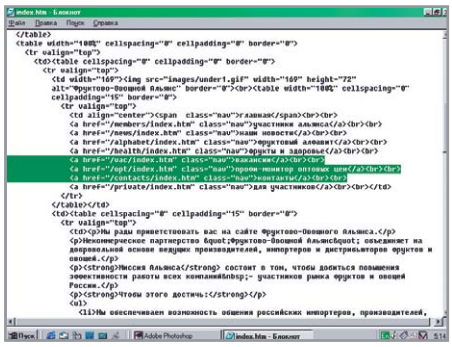
Как ни парадоксально, у статичного HTML есть значительные преимущества перед динамическими страницами. Не правда ли, крамольная мысль в век стремительного развития интерактивных технологий? Впрочем, судите сами.

Прежде всего, статические сайты обладают максимальной переносимостью — их можно безболезненно переместить с одного сервера на другой, если вдруг возникло желание по каким-либо соображениям сменить хостинг. При этом можно смело выбирать самые скромные тарифные планы — пусть провайдер запрещает какие угодно сервисы, но покажите мне такой хостинг, где не разрешены статичные HTML-страницы!

Еще один плюс заключается в том, что статичные документы можно смотреть



▲ Необходимо удалить ссылки на раздел «Профи-монитор оптовых цен» со всех страниц сайта



▲ Для выполнения задачи выделяем фрагмент HTML-кода, который требует замены

даже там, где вообще нет никакого сервера, то есть на любой локальной машине, на которой установлен лишь браузер. Это очень удобно, если хочется продемонстрировать очередной вариант оформления сайта непонятливому заказчику на его офисном компьютере, где «по недосмотру» не установлен Apache и откуда почему-то нет доступа в Интернет.

Нагрузка на web-сервер при использовании статичных HTML-страниц минимальна — ведь в данной ситуации сервер занят только тем, что выдает клиенту документ, имя которого запрошено.

Наконец, весьма существенным преимуществом «классики» является тот факт, что статичные HTML-файлы принципиально не могут причинить никакого вреда серверу, поскольку не предпринимают никаких активных действий, представляя собой нечто вроде «мертвого груза».

Видимый аскетизм

Так что же делать? Ограничиться «голым» HTML, отречься от всех достижений цивилизации и тратить свое драгоценное время на создание статичных страниц?!

Вовсе нет! Ведь никто не мешает перенести многие возможности, предоставляемые движком сайта, с далекого и туманного web-сервера на родную локальную машину, а на сервер закачивать модифицированные файлы по FTP в полностью готовом виде.

У такого подхода, кстати говоря, масса достоинств. Начнем с того, что он не требует постоянного подключения к Интернету. Ведь средства поддержки крупных сайтов, как правило, взаимодействуют с администратором при помощи некоего web-интерфейса. Выглядит это, спору нет, красиво, но хочешь — не хочешь, а будь добр в течение всего сеанса обновления сайта находиться в онлайне. Для специалиста, получающего фиксированный оклад в крупной компании, это, разумеется, не проблема, а вот фрилансеру, зарабатывающему деньги поддержкой сайтов и оплачивающему доступ в Интернет из своего собственного кармана, подобное решение вряд ли понравится. У такого человека каждая минута соединения с провайдером на счету.

А какие еще плюсы? Локальные скрипты могут быть очень простыми и незатейливыми, поскольку к ним не предъявляется ровным счетом никаких требований по безопасности и оптимизации кода. Эти сценарии пишутся каждым программистом под себя, стало быть, отпадает необходимость заботы о »

Альтернативные решения

Проще — значит удобнее

Концепция разделения содержания и представления, пропагандируемая Консорциумом W3C, позволяет коренным образом изменить дизайн сайта при минимальных затратах — в идеале нужно модифицировать единственный файл стилей. Но, к сожалению, в силу известных проблем совместимости реализовать такую идиллию в полной мере пока удается разработчикам весьма немногих web-проектов.

Решение поставленной проблемы с применением более привычных технологий заключается в использовании SSI (Server Side Includes, включения на стороне сервера). Фрагменты, которые потенциально могут быть подвержены глобальной замене (к примеру, строки HTML-кода, описывающие те или иные функциональные области страницы, будь то шапка, панель навигации или подвал), размещают в отдельных файлах. Директивы SSI оформляются как HTML-комментарии. Синтаксис одной из самых часто используемых директив SSI выглядит следующим образом: <!--#include virtual=«to_insert.html»-->. Если вставить такую строчку в тело любого HTML-документа (для определенности — index.html), то при условии, что на web-сервере разрешены SSI, эта запись будет динамически заменена содержимым файла to_insert.html перед выдачей запрошенного документа index.html клиенту. Так, на стороне сервера можно формировать HTML-страницы буквально «по кусочкам», ведь в одном файле может быть несколько директив SSI.

▲ Вставляем выделение в файлы old.txt и new.txt, удаляем ненужную строку из файла new.txt

▲ Дальнейшая работа — дело техники. Так выглядит экран при работе со скриптом ac.pl. Белые буквы на черном фоне, консольный режим, мечта сисадмина... Почему заменено 34 файла из 80, меньше половины? Потому что остальное приходится на английскую версию сайта и HTML-документы без навигации

» каком бы то ни было «товарном виде» скриптов. Во имя чего строить замысловатые регулярные выражения и тратить бесконечные строки кода на фильтрацию потока ввода, зачем ломать голову над тем, как сэкономить лишний килобайт оперативной памяти сервера и лишнюю долю секунды его процессорного времени? Согласитесь, ведь куда приятнее писать программки «школьного» уровня сложности, размышляя во время работы о чем-то возвышенном. Абсолютная свобода творчества — подумать только, ведь можно смело вызывать из скрипта любые команды оболочки операционной системы, стандартные утилиты или вообще программы без боязни, что нехороший дядя передаст им какой-нибудь опасный параметр!

От слов к делу

Набор скриптов, составляющих локальный движок того или иного сайта, может быть совершенно произвольным. Пожалуй, количество и назначение таких приложений зависит только от фантазии web-мастера и его лености. (Как известно, лень — двигатель прогресса, и вопрос, как она сказывается на качестве, к примеру, скриптов, весьма неоднозначен. Я склонен считать, что влияние сугубо положительное.)

Бывают и универсальные скрипты, пригодные для администрирования многих сайтов. Об одном из таких я и расскажу в этой статье. А в последующих материалах тему, связанную с «джентльменским набором» web-мастера, можно будет продолжить, поскольку она довольно обширна.

Одним из самых неблагодарных типов задач, встающих рано или поздно перед разработчиком любого web-проекта, является замена того или иного фрагмента кода на всех страницах сайта. Цели могут быть различными — от глобального изменения оформления до простой замены адреса электронной почты или номера телефона, на беду размещенного внизу каждой страницы, коих насчитывается порядка сотни.

Давайте же напишем скрипт, который бы делал эту рутинную работу за нас.

Итак, требуется создать сценарий, предназначенный для глобальной автозамены повторяющихся фрагментов HTML-кода во многих файлах сразу. Но если уж браться за создание средства автоматизации этого процесса, то нужно предусмотреть возможность замены блоков данных неограниченной длины. Предположим, мы полностью меняем дизайн сайта и нам нужно во всех HTML-страницах одним махом изменить 10-15 строк шапки и примерно столько же строк подвала.

Для решения задачи будем использовать великий и могучий язык Perl. Львиная доля CGI-скриптов, исполняющихся на серверах Web, как известно, написана именно на этом языке, по праву завоевавшем симпатии у многих web-разработчиков. Но код нашего сценария будет намного проще, чем у любого полноценного CGI-скрипта. Подробный листинг с комментариями вы можете увидеть в блоке «Пример скрипта».

Запуск движка

Теперь комментарии для пользователей. Для функционирования скрипта создайте два файла — old.txt и new.txt. В первый файл скопируйте без изменений фрагмент HTML-кода, который нужно заменить. Во второй скопируйте этот же самый фрагмент, после чего внесите в него необходимые коррективы прямо на месте. Сохраните оба файла и запустите скрипт на выполнение, указав в качестве параметра в командной строке путь к каталогу, в котором нужно искать HTML-документы для массовой замены. Если опустить параметр, будет использовано содержимое текущего каталога. При работе скрипта обрабатываются все файлы, соответствующие маске *.*htm* (то есть, например, все файлы с расширениями htm, html, shtml или xhtml), хранящиеся в указанной директории, включая подкаталоги. Скрипт рассчитан на работу под управлением Windows — он использует команду dir для получения списка файлов, которые нужно обработать.

Конечно, приведенный в качестве примера скрипт весьма прост. Подобный скрипт можно «научить» производить замену «не совсем одинаковых» блоков данных, заставить понимать регулярные выражения, вычислять какие-то значения в зависимости от чего-то и т. д. Насколько сложным вы захотите сделать движок сайта, какими средствами будете пользоваться — все это зависит от конкретных задач. Но использование подобных скриптов может существенно облегчить жизнь разработчику.

■ ■ ■ **Артемий Ломов**

▲ Результат работы: был раздел «Профи-монитор» — а теперь исчез за ненадобностью

Пример скрипта

Язык Perl — простое решение сложных задач

Устанавливаем счетчик времени в значение 0:

```
$starttime = time();
```

Назначение переменной $i будет пояснено ниже.

```
$i = 0;
```

Если файл old.txt существует, открываем его как поток OLD, определяем размер и читаем из него строку $old, длина которой равна размеру файла. Это строка, которую требуется заменить при помощи скрипта. В случае, если файл old.txt не найден, выводится сообщение и работа скрипта прекращается.

```
if (-e 'old.txt')
{
 @filestats = stat('old.txt');
 $filesize = $filestats[7];
 open(OLD, «old.txt»);
 sysread (OLD, $old, $filesize);
 close(OLD);
}
else
{
 print «AutoChange: Не удается обнаружить файл old.txt.\n»;
 exit(0);
}
```

Аналогичным образом из файла new.txt считывается строка $new, представляющая собой текст, на который требуется заменить фрагменты HTML-документов, совпадающие со значением переменной $old.

```
if (-e 'new.txt')
{
 @filestats = stat('new.txt');
 $filesize = $filestats[7];
 open(NEW, «new.txt»);
 sysread (NEW, $new, $filesize);
 close(NEW);
}
else
{
 print «AutoChange: Не удается обнаружить файл new.txt.\n»;
 exit(0);
}
```

Считываем параметр, заданный в командной строке, и присваиваем его переменной $path, обозначающей путь к каталогу с HTML-файлами.

```
$path = shift(@ARGV);
```

Если путь — не пустая строка и на конце его нет обратного слэша, добавляем этот самый слэш в конец пути.

```
if (($path ne '') && (substr($path, -1) ne '\\'))
{
 $path = $path . '\\';
}
```

Используем возможности командной среды операционной системы — запускаем команду dir с ключами /l (использование нижнего регистра), /s (вывод списка файлов из заданной папки и ее подпапок) и /b (вывод только имен файлов без указания размеров, дат и времени создания). Осуществляем перенаправление вывода в файл filelist.txt.

```
open (FILELIST, «| dir $path\*\.\*htm\* /l /s /b -> filelist.txt»);
close (FILELIST);
```

При условии существования файла filelist.txt определяем его размер. Если размер нулевой или файл вовсе отсутствует, выводим сообщение «Нет файлов для обработки» и завершаем выполнение скрипта. В случае обнаружения всех необходимых данных открываем filelist.txt для чтения как поток FILELIST. Поток с таким именем уже использовался ранее, но он был закрыт, так что повторное применение этого имени не приведет к ошибке.

```
if (-e 'filelist.txt')
{
 @filestats = stat('filelist.txt');
 $filesize = $filestats[7];
 if ($filesize == 0)
 {
  print «AutoChange: Нет файлов для обработки.\n»;
  exit(0);
 }
 open (FILELIST, «filelist.txt»);
}
else
{
 print «AutoChange: Не удается обнаружить файл filelist.txt.\n»;
 exit(0);
}
```

Считываем названия файлов из потока FILELIST, пока он не иссякнет. Каждый раз записываем считанное имя в переменную $filename. При существовании файла с именем, совпадающим со строкой $filename, открываем его для чтения как поток CURFILE, считывая содержимое и записывая его в переменную $filedata. Переменная $chflag используется как флаг (изменен ли текущий файл, 1 — да, 0 — нет). Изменение производится только в том случае, если значение строки $filedata содержит в себе подстроку, эквивалентную значению $old, хотя бы один раз. Подстрока, соответствующая $old, в строке $filedata заменяется на $new, и новая строка $filedata записывается в новый поток с тем же самым именем CURFILE. В данном случае поток CURFILE открыт для записи в файл $filename. Переменная $i используется как счетчик обновленных файлов. Счетчик $j отражает количество обработанных файлов в целом. Переменная $curpos используется как индикатор текущей позиции при обработке строки $filedata.

```
while ($filename = <FILELIST>)
{
 chomp ($filename);
 if (-e $filename)
 {
  @filestats = stat($filename);
  $filesize = $filestats[7];
  open (CURFILE, «$filename»);
  sysread (CURFILE, $filedata, $filesize);
  close (CURFILE);
  $chflag = 0;
  $curpos = 0;
  while (index($filedata, $old, $curpos) >= 0)
  {
   $filedata = substr($filedata, 0, index($filedata, $old,
   $curpos)) . $new . substr($filedata, index($filedata, $old,
   $curpos) + length($old));
   $curpos = $curpos + length($new);
   $chflag = 1;
  }
  if ($chflag == 1)
  {
   open (CURFILE, «>$filename»);
   syswrite (CURFILE, $filedata, length($filedata));
   close (CURFILE);
   print «AutoChange: изменен файл $filename\n»;
   $i++;
  }
  $j++;
 }
 else
 {
  print «AutoChange: Не удается найти файл $filename\n»;
 }
}
close (FILELIST);
$jobtime = time() — $starttime;
print «AutoChange: Обработано $j файлов, из них обновлено: $i\.\n»;
print «Общее время работы составило $jobtime секунд.\n»;
if ($j -> 0)
{
 print «Список всех обработанных файлов смотрите в filelist.txt.\n»;
}
exit(0);
```