libwww-perl

LWP – библиотека для доступа к WWW из Perl

Версия 5.48

Перевод: Вадим Хомаха, © 2001

ПРЕДИСЛОВИЕ
автора перевода

Данная публикация посвящена наиболее темному, но самому нужному для продвинутого разработчика web-приложений аспекту программирования на языке Perl – автоматизации задач для World Wide Web и созданию Web – роботов.

Честно говоря, я никогда бы не подумал, что для любимого мною модуля языка Perl может быть такая корявая документация ☺ . Насколько я привык доверять документации, но в случае LWP нужно сразу же лезть в исходный текст модулей. Только после этого в голове наступает некоторое прояснение (не всегда!).

Конечно же, чтение этой документации требует серьезной предварительной подготовки. В частности, требуется хорошее знание нюансов протокола HTTP (спецификации RFC2616), CGI, правил работы с роботами и многого другого.

Джисл Аас, автор LWP – libwww-perl, - проделал огромную работу, облегчив задачу автоматизации задач Web, но с английскимя зыком у него, к сожалению, большие проблемы. Иногда очень трудно было понять, что же, в конце концов, он хотел сказать… Отсюда и качество учебника – поваренной книги LWP в примерах… и документации ко всем прочим модулям пакета.

Несмотря на низкое качество документации, сама реализация библиотеки достойна восхищения. Создать нормального робота, написав десяток строк на Perl – это просто фантастика! А приложение – полнофункциональный сервер HTTP в виде пары дюжин строк?

Я не претендую на роль истины в последней инстанции. Человеку свойственно ошибаться – увы! Качество моего перевода, наверняка, не лучше, чем большинства у книг, имеющихся в продаже. Причин – несколько, основная – отсутствие

Попытка – не пытка, сказал палач. Читайте, критикуйте, присылайте свои пожелания и толкования. Все они будут приняты с благодарностью.

Всегда Ваш,

Вадим Хомаха

Февраль 2001

Терминология

С точки зрения правильности восприятия следующего материала необходимо предварительно определиться с терминами. Данный раздел не является частью документации LWP, а лишь переводом официальной спецификации действующего протокола HTTP1.1 (RFC-2616) . Данный раздел добавлен переводчиком. Перечисленные ниже термины приведены в порядке, отличающемся от приведенного в спецификации RFC-2616.

В спецификации RFC-2616 определены следующие термины:

server - сервер Приложение, которое слушает соединения, принимает запросы на обслуживание и посылает ответы. Любая программа такого типа способна быть как клиентом, так и сервером; наше использование данного термина относится скорее к роли, которую программа выполняет, создавая некоторые соединения, нежели к возможностям программы вообще. Аналогично, любой сервер может функционировать в разных режимах (включая режим прокси-сервера), изменяя поведение, основываясь на характере каждого запроса.

client - клиент Программа, которая устанавливает соединения с целью отправки запросов.

user agent – пользовательский агент Клиент, инициирующий запрос. Как правило, браузеры, редакторы, роботы (spiders), или другие инструменты конечного пользователя.

proxy прокси-сервер Программа-посредник, которая действует и как сервер, и как клиент с целью создания запросов от имени других клиентов. Запросы обслуживаются прокси-сервером, или передаются им, возможно, с изменениями. Прокси-сервер должен удовлетворять требованиям клиента и сервера, согласно спецификации rfc2616.

connection - соединение Виртуальный канал транспортого уровня, установленный между двумя программами с целью связи

request - запрос Любое сообщение HTTP, содержащее запрос.

response – отклик, ответ Любое сообщение HTTP, содержащее ответ

fresh - свежий Отклик считается свежим, если его возраст еще не превысил время жизни.

resource - ресурс Сетевой объект данных или сервис, который может быть идентифицирован URI. Ресурсы могут быть доступны в нескольких представлениях (например, на нескольких языках, в разных форматах данных, иметь различный размер, разрешающую способность) или различаться по другим параметрам.

message – сообщение Основной модуль HTTP связи, состоящей из структурной последовательности октетов, соответствующих синтаксису, определенному в разделе 4 спецификации rfc2616 и передаваемых по соединению.

entity - объект Информация, передаваемая в качестве полезной нагрузки запроса или отклика. Объект состоит из метаинформации в форме полей заголовка объекта и содержимого (контента) в форме тела объекта.

variant - вариант Ресурс может иметь одно, или несколько представлений, связанных с ним в данный момент. Каждое из этих представлений называется "вариант".

content negotiation – обсуждение содержимого Механизм для выбора соответствующего представления во время обслуживания запроса, как описано в разделе 12 rfc2616. может обсуждаться представление объектов в любом отклике (включая ошибочные ответы).

representation - представление Объект, включенный в ответ, и подчиняющийся обсуждению содержимого (Content Negotiation). Может существовать несколько представлений, связанных со специфическими состояниями отклика.

llibwww-perl в примерах (поваренная книга LWP)

Данный документ содержит несколько примеров типичного использования библиотеки libwww-perl. Для более детального ознакомления обратитесь к документации по отдельным модулям.

Все приведенные примеры должны быть исполняемыми программами. В большинстве случаев Вы можете протестировать приведенный код, указав имя файла в строке вызова интерпретатора perl.

GET

Самым легким примером использования данной библиотеки является получение документов из сети. Модуль LWP::Simple обеспечивает функцию get(), которая возвращает документ, определенный URL, переданным ей в качестве параметра:

```
use LWP::Simple;
$doc = get 'http://www.sn.no/libwww-perl/';
```

или, в виде однострочной программы Perl, напечатать документ в поток стандартного вывода при помощи функции getprint():

```
perl -MLWP::Simple -e 'getprint "http://www.sn.no/libwww-perl/";'
```

или, выкачать самую свежую версию Perl путем выполнения следующей команды:

```
perl -MLWP::Simple -e '
  getstore "ftp://ftp.sunet.se/pub/lang/perl/CPAN/src/latest.tar.gz";,
           "perl.tar.gz"'
```

А, возможно, Вы захотите сначала найти одно из ближайших к Вам зеркал CPAN, выполнив следующую команду:

```
perl -MLWP::Simple -e 'getprint "http://www.perl.com/CPAN-local/MIRRORED.BY";'
```

Однако хватит возиться с простыми примерами! Объектно-ориентированный интерфейс LWP предоставляет Вам гораздо большие возможности управления запросом, отправленным серверу. Используя этот интерфейс, Вы можете полностью контролировать процесс формирования заголовков отправляемых запросов, а также способы обработки возвращаемых откликов.

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$ua->agent("$0/0.1 " . $ua->agent);
# $ua->agent("Mozilla/8.0") # Притворимся совместимым браузером ☺
$req = HTTP::Request->new(GET => 'http://www.linpro.no/lwp');
$req->header('Accept' => 'text/html');
# send request
$res = $ua->request($req);
# check the outcome
if ($res->is_success) {
   print $res->content;
} else {
   print "Error: " . $res->status_line . "\n";
}
```

Программа lwp-request (известная под псевдонимом GET), распространяемая вместе с библиотекой, также может быть использована для выкачивания документов с серверов WWW.

HEAD

Если Вы хотите проверить только наличие документа (т.е. проверить правильность URL), попытайтесь воспользоваться таким фрагментом кода:

```
use LWP::Simple;
if (head($url)) {
   # Отлично -  Документ существует
}
```

Функция head() в действительности возвращает meta – информацию о документе в виде списка. Первые три элемента возвращаемого списка - это тип документа, размер и возраст документа.

Больший контроль над запросом или доступ ко всем полям заголовка полученного отклика требует использования объектно-ориентированного интерфейса, описанного выше для GET. Просто замените в нужных местах GET на HEAD - s/GET/HEAD/g ☺.

POST

Не существует процедурного интерфейса для передачи данных на WWW сервер. Вы должны воспользоваться для этого объектно-ориентированным интерфейсом. Наиболее часто POST – операция используется для доступа к приложению Web-форме:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
my $req = HTTP::Request->new(POST => 'http://www.perl.com/cgi-bin/BugGlimpse');
$req->content_type('application/x-www-form-urlencoded');
$req->content('match=www&errors=0');
my $res = $ua->request($req);
print $res->as_string;
```

Ленивые программисты используют модуль HTTP::Request::Common to для передачи надлежащего сообщения - запроса методом POST (он хорошо обрабатывает все необходимые esc-последовательности) и имеет подходящее значение по умолчанию для content_type:

```
use HTTP::Request::Common qw(POST);
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
my $req = POST 'http://www.perl.com/cgi-bin/BugGlimpse',
                [ search => 'www', errors => 0 ];
print $ua->request($req)->as_string;
```

Программа lwp-request (известная под псевдонимом POST), которая распространяется вместе с библиотекой, также может быть использована для передачи запросов методом POST.

Прокси-серверы

Некоторые сайты используют прокси-серверы для того, чтобы получить доступ к информации, размещенной за брандмауэром или просто в качестве кэша для улучшения производительности. Прокси – серверы также могут использоваться для

доступа к ресурсам по протоколам, которые не поддерживаются непосредственно (или поддерживаются недостаточно хорошо ☺) библиотекой libwww-perl.

Вы можете присвоить начальные значения параметрам настройки доступа к прокси - серверу перед отправкой запроса:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$ua->env_proxy; # инициализация из переменной окружения
# или
$ua->proxy(ftp  => 'http://proxy.myorg.com');
$ua->proxy(wais => 'http://proxy.myorg.com');
$ua->no_proxy(qw(no se fi));
my $req = HTTP::Request->new(GET => 'wais://xxx.com/');
print $ua->request($req)->as_string;
```

Интерфейс LWP::Simple вызовет env_proxy() для Вас автоматически. Приложения, использующие метод $ua->env_proxy(), обычно не используют методы $ua->proxy() и $ua->no_proxy().

Некоторые прокси - серверы также требуют, чтобы Вы передали им имя пользователя и пароль для разрешения прохождения через них запроса. Вы можете это сделать, добавив соответствующий заголовок, например, так:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$ua->proxy(['http', 'ftp'] => 'http://proxy.myorg.com');
$req = HTTP::Request->new('GET',"http://www.perl.com";);
$req->proxy_authorization_basic("proxy_user", "proxy_password");
$res = $ua->request($req);
print $res->content if $res->is_success;
```

Замените proxy.myorg.com, proxy_user и proxy_password подходящими значениями для Вашего сайта.

Доступ к защищенным документам

Документы, защищенные при помощи базовой авторизации, могут быть получены с легкостью следующим образом:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$req = HTTP::Request->new(GET => 'http://www.linpro.no/secret/');
$req->authorization_basic('aas', 'mypassword');
print $ua->request($req)->as_string;
```

Альтернативой может стать введение производного класса *LWP::UserAgent,* который подменит метод get_basic_credentials(). Изучите программу *lwp-request* в качестве примера такой реализации.

Файлы COOKIE

Некоторые сайты обожают играться с файлами cookie. По умолчанию LWP игнорирует файлы cookies, которые отправляют посещаемые серверы. Однако Вы можете разрешить их получение, передав новый объект HTTP::Cookies методу cookie_jar.

```
use LWP::UserAgent;
use HTTP::Cookies;
$ua = LWP::UserAgent->new;
```

```
$ua->cookie_jar(HTTP::Cookies->new(file => "lwpcookies.txt",
                                   autosave => 1));
# а затем посылаете запросы как Вы это обычно делаете
$res = $ua->request(HTTP::Request->new(GET => "http://www.yahoo.no";));
print $res->status_line, "\n";
```

По мере того, как Вы будете посещать сайты, посылающие Вам файлы cookie, файл "*lwpcookies.txt*" будет разрастаться.

HTTPS

Документы с адресами URL, доступные по протоколу https, могут быть получены точно таким же образом, как и при доступе по протоколу http, при условии, что интерфейсный модуль SSL был установлен надлежащим образом (читайте файл *README.SSL* из дистрибутивного пакета libwww-perl для получения полной информации). Если интерфейс SSL для LWP не установлен, то вы получите сообщение об ошибке "501 Protocol scheme 'https' is not supported (Ошибка 501- Протокол https не поддерживается)" при попытке получения документа с таким адресом URL.

Ниже приведен пример получения и печати Web-страницы с использованием SSL:

```
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
my $req = HTTP::Request->new(GET => 'https://www.helsinki.fi/');
my $res = $ua->request($req);
if ($res->is_success) {
    print $res->as_string;
} else {
    print "Failed: ", $res->status_line, "\n";
}
```

Создание зеркал

Если Вы хотите создать зеркальные копии некоторых документов с сервера WWW, попытайтесь выполнить следующий фрагмент кода через регулярные промежутки времени:

```
use LWP::Simple;
%mirrors = (
   'http://www.sn.no/'              => 'sn.html',
   'http://www.perl.com/'           => 'perl.html',
   'http://www.sn.no/libwww-perl/' => 'lwp.html',
   'gopher://gopher.sn.no/'         => 'gopher.html',
);
while (($url, $localfile) = each(%mirrors)) {
   mirror($url, $localfile);
}
```

Или, в виде однострочной Perl - программы:

```
perl -MLWP::Simple -e 'mirror("http://www.perl.com/";, "perl.html")';
```

Документ не будет передан до тех пор, пока он не будет обновлен.

Документы большого размера

Если размер получаемого документа слишком велик, и он не помещается в оперативной памяти, у Вас имеется два альтернативных решения этой проблемы.

Вы можете отдать библиотеке распоряжение записать содержимое полученного документа в файл (в этом случае вторым аргументом метода $ua->request() будет имя файла):

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
my $req = HTTP::Request->new(GET =>
                'http://www.linpro.no/lwp/libwww-perl-5.46.tar.gz');
$res = $ua->request($req, "libwww-perl.tar.gz");
if ($res->is_success) {
   print "ok\n";
}
else {
   print $res->status_line, "\n";
}
```

Или Вы можете обрабатывать документ по мере того, как он прибывает (вторым аргументом $ua->request() будет ссылка на фрагмент кода). Например:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$URL = 'ftp://ftp.unit.no/pub/rfc/rfc-index.txt';
my $expected_length;
my $bytes_received = 0;
my $res =
   $ua->request(HTTP::Request->new(GET => $URL),
              sub {
                  my($chunk, $res) = @_;
                  $bytes_received += length($chunk);
                  unless (defined $expected_length) {
                     $expected_length = $res->content_length || 0;
                  }
                  if ($expected_length) {
                       printf STDERR "%d%% - ",
                                 100 * $bytes_received / $expected_length;
                  }
                  print STDERR "$bytes_received байт получено\n";
                  # Обработать каким-либо образом полученную порцию данных
                  # print $chunk;
              });
  print $res->status_line, "\n";
```

Рассмотрим подробнее основной модуль библиотеки.

LWP - Библиотека для доступа к WWW из Perl. Версия 5.48

Наименование

LWP – libwww-perl – библиотека для доступа к WWW из Perl

Поддерживаемые платформы

- Linux
- Solaris
- Windows

Синтаксис

```
use LWP;
print "Это версия библиотеки libwww-perl-$LWP::VERSION\n";
```

Описание

Libwww-perl – это собрание модулей Perl, которые обеспечивают простой и непротиворечивый программный интерфейс (API) к WWW. Основная цель библиотеки – обеспечить классы и функции, которые позволяют Вам писать программы-клиенты

WWW, таким образом, libwww-perl – это клиентская библиотека WWW. Библиотека также содержит функции и более общего назначения.

Большинство модулей библиотеки – объектно-ориентированны. Пользовательский агент, посланный запрос и отклик, полученный от WWW-сервера, представлены объектами. Это позволяет реализовать простой и мощный интерфейс к сервисам Интернет. Данный интерфейс можно легко расширить и приспособить к Вашим требованиям.

Основные возможности, реализуемые библиотекой:

- Библиотека содержит различные компоненты (модули), которые могут быть использованы совместно или по отдельности.
- Обеспечивает объектно-ориентированную модель обмена информацией в стиле HTTP. В рамках этой модели библиотека поддерживает доступ к ресурсам http, https, gopher, ftp, news, file, и mailto.
- Обеспечивает полностью объектно-ориентированный или простой процедурный интерфейс.
- Поддерживает базовую и цифровую схемы авторизации (контроля доступа).
- Поддерживает прозрачную обработку редиректов (перенаправление запросов).
- Поддерживает доступ через прокси-серверы.
- Обеспечивает синтаксический анализатор файлов *robots.txt* и механизм конструирования роботов.
- Взаимодействует с Tk. Простой графический браузер, основанный на Tk-технологии, который называется 'tkweb' распространяется с расширением Tk для perl.
- Обеспечивает алгоритм согласования содержимого HTTP, который может быть использован и в модулях поддержки протокола, и в серверных сценариях (к примеру, сценариях CGI).
- Поддерживает файлы HTTP cookie.
- Включает в себя простое приложение, запускаемое из командной строки, - `lwp-request`.

Модель обмена информацией в стиле HTTP

Библиотека libwww-perl основана на модели обмена информацией в стиле HTTP. В этом разделе мы попытаемся объяснить, что это означает.

Позвольте начать со следующей цитаты из официальной спецификации HTTP http://www.w3.org/pub/WWW/Protocols/:

> Протокол HTTP основан на принципе запрос-ответ. Клиент устанавливает соединение с сервером и посылает запрос к серверу в форме: метода запроса, URI, и версии используемого протокола, за которыми следует сообщение в кодировке MIME, содержащее модификаторы запроса, информацию о клиенте, и, возможно, тело сообщения. Сервер отправляет обратно строку состояния, которая состоит из версии протокола, использованной для отправки сообщения, код успешного завершения или ошибки, за которым следует собственно сообщение в кодировке MIME. В сообщении содержится информация о сервере, существенная информация из META-тэгов, и, возможно, содержимое сообщения, заключенное между тэгами <BODY>.

Для библиотеки libwww-perl это означает, что обмен с сервером всегда осуществляется в следующей последовательности: Сначала создается и конфигурируется объект - запрос *request*. Затем этот объект передается серверу, и мы в ответ получаем объект -

отклик *response,* который мы можем обработать. Запрос всегда независим от всех предыдущих запросов, т.е. не сохраняет информацию о предыдущем состоянии. Одна и та же простая модель используется для любого вида сервиса, к которому мы хотим обратиться.

Например, если мы хотим получить определенный документ с сервера, мы посылаем ему запрос, в котором содержится имя запрашиваемого документа, а отклик содержит сам требуемый документ. Если мы обращаемся к поисковой машине, то содержимое запроса будет содержать параметры запроса, а отклик – результат поиска в базе поисковой машины. Если мы хотим отправить кому-либо сообщение по электронной почте, то объект – запрос к почтовому серверу будет содержать наше сообщение, а объект – отклик будет содержать подтверждение того, что сообщение было принято и переправлено отправителю (отправителям).

Не правда ли, очень просто?!

Объект - запрос

Объект – запрос libwww-perl представлен классом с именем `HTTP::Request`. То, что в имени класса используется префикс `HTTP::`, фактически только подразумевает, что мы используем модель обмена информацией в стиле HTTP. Она не ограничивает виды сервисов, которые мы можем передать объекту - *запросу*. Например, мы можем отправлять запросы `HTTP::Request` как к ftp, так и к gopher серверам, а также к локальной файловой системе.

Главные свойства объектов-запросов:

- **method** представляет собой короткую строку, которая объявляет используемый тип запроса. Самыми популярными методами являются **GET**, **PUT**, **POST** и **HEAD**.
- **url** - это строка, указывающая протокол, сервер и имя "документа", к которому мы хотим получить доступ. **url** может также включать в себя также различные другие параметры.
- **headers** содержит дополнительную информацию о запросе, а также может использоваться для описания содержимого запроса. Заголовки отсылаются в виде пар ключ=значение.
- **content** - произвольное количество данных.

Объект-отклик

Объект-отклик библиотеки libwww-perl представлен классом `HTTP::Response`. Основные свойства объектов данного класса:

- **code** – числовой код, который обозначает результат запроса.
- **message** –короткое доступное для восприятия пользователя сообщение, соответствующее коду *code*.
- **headers** – заголовки, содержащие дополнительную информацию об отклике и описывающие содержимое отклика.
- **content** – произвольное количество данных.

В связи с тем, что мы не желаем хранить все возможные значения кодов завершения *code* непосредственно в наших программах, объект – отклик libwww-perl содержит методы, которые могут быть использованы для проверки типа запроса. Наиболее часто используемыми методами классификации откликов являются:

`is_success()`
: Запрос был успешно получен, понят или принят.

`is_error()`
: Запрос не успешен. Сервер или ресурс может быть недоступен, доступ к ресурсу может быть запрещен, а также по некоторым другим причинам.

Пользовательский агент (User Agent)

Предположим, что мы создали объект-*запрос*. Что нужно реально с ним сделать, чтобы получить ответ - *отклик*?

Ответ состоит в том, чтобы передать объект – запрос объекту – *пользовательскому агенту* и уже этот агент принимает на себя заботу обо всех действиях, которые нужно выполнить (подобно низкоуровневому обмену данными и обработке ошибок) и возвращает объект - *отклик*. Пользовательский агент представляет Ваше приложение в сети и предоставляет Вам интерфейс, принимающий *запросы* и возвращающий *отклики*.

Пользовательский агент – это интерфейсный уровень между кодом Вашего приложения и сетью. Вы можете получить доступ к различным серверам сети именно посредством этого интерфейса.

Имя класса пользовательского агента библиотеки libwww-perl - `LWP::UserAgent`. Каждое приложение libwww-perl, которое хочет установить связь с Сетью, должно создать, по крайней мере, один экземпляр объекта этого класса. Основной метод, обеспечиваемый этим объектом - request(). Этот метод в качестве аргумента принимает объект `HTTP::Request` и (в конечном счете) возвращает объект `HTTP::Response`.

Пользовательский агент имеет множество других атрибутов, которые позволяют Вам настроить его поведение и характер его взаимодействия с сетью и с Вашим приложением.

- **timeout** определяет интервал времени, в течение которого удаленный сервер должен ответить на запрос, прежде чем произойдет отсоединение и библиотека сгенерирует внутреннее событие-отклик *timeout*.
- **agent** определяет имя, используемое Вашим приложением для представления себя в сети.
- Атрибуту **from** может быть присвоено значение адреса e-mail лица, ответственного за функционирование данного приложения. Если данный параметр определен, то этот адрес будет отправлен каждому серверу в каждом запросе.
- **parse_head** указывает, нужно ли инициализировать заголовки отклика информацией секции <head> документов HTML.
- Атрибуты **proxy** и **no_proxy** определяют, нужно ли, и каким образом передавать запрос прокси – серверу. Смотрите подробное описание в документе http://www.w3.org/pub/WWW/Proxies/ .
- Атрибут **credentials** обеспечивает способ указания имени пользователя и паролей, необходимых для доступа к некоторым сервисам.

Многие приложения нуждаются в большем контроле над процессом взаимодействия с сетью; они могут достичь этого путем создания наследуемого класса `LWP::UserAgent`. Библиотека включает в себя производный класс `LWP::RobotUA` для организации приложений - роботов.

Пример

Данный пример показывает, как объекты – пользовательский агент, запрос и ответ, - реализуются в реальном коде Perl:

```
# Создать  новый объект – пользовательский агент
use LWP::UserAgent;
$ua = new LWP::UserAgent;
$ua->agent("AgentName/0.1 " . $ua->agent);
# Создать запрос
my $req =
   new HTTP::Request POST => 'http://www.perl.com/cgi-bin/BugGlimpse';
$req->content_type('application/x-www-form-urlencoded');
$req->content('match=www&errors=0');
# Передать запрос пользовательскому агенту и получить ответ – отклик
my $res = $ua->request($req);
# Check the outcome of the response
if ($res->is_success) {
    print $res->content;
} else {
    print "На этот раз Вас постигла неудача\n";
}
```

Пользовательский агент $ua создается только один раз – при запуске приложения. Для каждого посылаемого запроса должен быть создан отдельный объект-запрос.

Межсетевой уровень

В этом разделе обсуждаются различные протоколы доступа, а также методы HTTP, чьи заголовки могут быть использованы для каждого из протоколов.

Ко всем запросам добавляется заголовок "User-Agent", который инициализируется атрибутом $ua->agent перед передачей запроса на межсетевой уровень. Подобным же способом заголовок "From" инициализируется атрибутом $ua->from.

Ко всем откликам библиотека добавляет заголовок с именем "Client-Date". Этот заголовок содержит время, когда отклик был получен Вашим приложением. Формат и семантика заголовка аналогичны формату и семантике заголовка "Date", создаваемого сервером. Вам могут также встретиться другие заголовки типа "Client-XXX". Все они сгенерированы внутренними процедурами библиотеки, а не получены от серверов, которым отправлены запросы.

Запросы HTTP

Запросы HTTP передаются серверу HTTP, и уже он решает, что делать с поступившим запросом. Некоторые серверы обрабатывают не только стандартные методы "GET", "HEAD", "POST" и "PUT", однако сценарии CGI могут обеспечивать реализацию любых методов.

Если сервер не доступен, библиотека сгенерирует внутреннее сообщение об ошибке.

Библиотека автоматически добавляет поля заголовка "Host" и "Content-Length" к HTTP запросу перед отправкой его по сети.

Вы можете добавить заголовок "If-Modified-Since" к запросу GET, для того , чтобы ввести дополнительное условие в запрос.

Для запросов POST Вы можете добавить заголовок "Content-Type". При эмулировании поведения формы в документе HTML <FORM> Вам, обычно, потребуется присвоить значение "application/x-www-form-urlencoded" заголовку "Content-Type". Описание примеров смотрите в Поваренной книге LWP.

Текущая реализация HTTP – запросов библиотеки libwww-perl поддерживает протокол HTTP/1.0. Серверы, работающие по протоколу HTTP/0.9, также обрабатываются корректно.

Библиотека позволяет получить доступ к прокси-серверам при помощи протокола HTTP. Это означает, что Вы можете настроить библиотеку таким образом, чтобы пересылать запросы всех типов при помощи модуля протокола HTTP. Подробное описание смотрите в руководстве LWP::UserAgent.

Запросы HTTPS

Запросы HTTPS это запросы HTTP, переданные по зашифрованному соединению при помощи протокола SSL, разработанного Netscape. Все перечисленное выше для запросов HTTP полностью справедливо и для запросов HTTPS. В дополнение к обычным заголовкам библиотека добавляет к отклику заголовки "Client-SSL-Cipher", "Client-SSL-Cert-Subject" и "Client-SSL-Cert-Issuer". Эти заголовки объясняют используемый метод шифрования и определяют имя владельца сервера.

Запрос может содержать заголовок "If-SSL-Cert-Subject" для того, чтобы сделать запрос зависимым от содержимого сертификата сервера. Если сертификат не соответствует ожидаемому, запрос к серверу не отсылается, и возвращается сообщение об ошибке, сгенерированное внутри библиотеки. Значение заголовка “If-SSL-Cert-Subject” интерпретируется в качестве регулярного выражения Perl.

Запросы FTP

В настоящее время библиотека поддерживает запросы GET, HEAD и PUT. GET получает список файлов и директорий с сервера FTP. PUT сохраняет файл на сервере FTP.

Вы можете указать параметры учетной записи ftp для тех серверов, которые требуют эту информацию помимо имени пользователя и пароля, включив в запрос заголовок "Account".

Имя пользователя и пароль можно указать при помощи основной процедуры авторизации, либо закодировав их в URL. Неуспешные входы на сервер ftp возвращают отклик UNAUTHORIZED (ДОСТУП ЗАПРЕЩЕН) с заголовком “WWW-Authenticate: Basic” и могут быть истолкованы аналогично основной схеме авторизации доступа HTTP.

Библиотека поддерживает символьный (ASCII) режим передачи файлов путем указания параметра "type=a" в строке URL.

Перечни файлов в каталогах по умолчанию возвращаются необработанными (в виде, полученном с сервера ftp),а типом принятых данных будет "text/ftp-dir-listing". Модуль `File::Listing` реализует методы для обработки таких перечней файлов.

Модуль ftp также может преобразовывать перечни файлов в каталогах в документы HTML, которые могут быть запрошены с помощью стандартного механизма обсуждения содержимого - выбора наиболее подходящего варианта (для этого Вы должны добавить в запрос заголовок “Accept: text/html”).

Для того, чтобы загрузить файл с сервера в обычном режиме, предполагается что значение “Content-Type» основано на расширении файла. Смотрите подробности в руководстве LWP::MediaTypes.

Заголовок запроса "If-Modified-Since" работает для серверов, на которых реализована команда MDTM, хотя, возможно, и не работает для перечней файлов в каталогах.

Пример:

```
$req = HTTP::Request->new(GET => 'ftp://me:passwd@ftp.some.where.com/');
$req->header(Accept => "text/html, */*;q=0.1");
```

Запросы News

Доступ к системе новостей USENET реализован протоколом NNTP. Имя сервера новостей берется из переменной окружения NNTP_SERVER; по умолчанию ему присвоено значение "news". Невозможно указать имя NNTP сервера новостей в виде news:URL.

Библиотека поддерживает методы GET и HEAD для получения статей сервера новостей с помощью протокола NNTP. Вы также можете отправлять статьи в группы новостей с помощью (сюрприз!) метода POST.

Тип запроса GET для запроса к группам новостей пока не реализован.

Примеры:

```
$req = HTTP::Request->new(GET => 'news:abc1234@a.sn.no');
$req = HTTP::Request->new(POST => 'news:comp.lang.perl.test');
$req->header(Subject => 'Проверка',
             From    => 'me@some.where.org');
$req->content(<<EOT);
Это текст сообщения, которое мы собираемся отправить во внешний мир
EOT
```

Запросы Gopher

Библиотека поддерживает методы GET и HEAD для запросов gopher. Все значения полей из заголовка запроса игнорируются. Метод HEAD мошенничает и возвращает отклик, даже не обращаясь к серверу.

Меню gopher всегда преобразовываются в HTML.

Отклик "Content-Type" генерируется на основе типа документа, закодированного (в качестве первой буквы) в самом запрашиваемом пути URL.

Пример:

```
$req = HTTP::Request->new(GET => 'gopher://gopher.sn.no/');
```

Файловые запросы

Библиотека поддерживает методы GET и HEAD для файловых запросов. Обеспечивается поддержка заголовка "If-Modified-Since". Все другие заголовки игнорируются. Компонент *host* адреса URL запрашиваемого файла должен быть либо пустым, либо иметь значение "localhost". Любое другое значение *host* будет признано ошибкой.

Директории всегда преобразовываются в документы HTML. Для обычных файлов заголовки отклика "Content-Type" и "Content-Encoding" определяются на основе расширений (типов) файлов.

Пример:

```
$req = HTTP::Request->new(GET => 'file:/etc/passwd');
```

Запросы Mailto

Вы можете отправлять (другими словами, "POST"ить) сообщения по электронной почте с помощью данной библиотеки. Все заголовки, указанные в запросе, передаются почтовой системе. Поле заголовка "To" инициализируется из адреса электронной почты в URL.
Пример:

```
$req = HTTP::Request->new(POST => 'mailto:libwww@perl.org');
$req->header(Subject => "subscribe");
$req->content("Подпишите меня, пожалуйста, на список рассылки libwww-perl!\n");
```

Обзор классов и пакетов

Данная таблица дает краткий обзор классов, реализуемых библиотекой. Отступ означает наследование.

```
LWP::MemberMixin   -- Доступ к внутренним переменным классов Perl5
  LWP::UserAgent   -- класс пользовательского агента WWW
    LWP::RobotUA   -- Класс для разработки приложений - роботов
  LWP::Protocol          -- Интерфейс к различным протоколам
    LWP::Protocol::http  -- доступ к документам http://
    LWP::Protocol::file  -- доступ к локальным файлам file://
    LWP::Protocol::ftp   -- доступ к файлам на ftp:// сервере
    ...
LWP::Authen::Basic – Обработка запросов с кодами 401 и 407
LWP::Authen::Digest  (аутентификация доступа пользователя)
HTTP::Headers      -- сообщение типа MIME/RFC822 (используется модулем
                      HTTP::Message)
HTTP::Message      -- сообщение в стиле HTTP
  HTTP::Request    -- Запрос HTTP
  HTTP::Response   -- Отклик HTTP
HTTP::Daemon       -- Класс для реализации сервера HTTP
WWW::RobotRules    -- Анализ файлов robots.txt
  WWW::RobotRules::AnyDBM_File – Сохранение правил RobotRules в файле
```

Следующие модули реализуют различные функции и определения:

```
LWP                -- Основной модуль. Номер версии библиотеки и
                      документация.
LWP::MediaTypes    -- Конфигурация типов MIME (text/html etc.)
LWP::Debug         -- Модуль протоколирования отладки
LWP::Simple        -- Упрощенный процедурный интерфейс для основных функций
HTTP::Status       -- Коды состояния HTTP (200 - OK и т.д.)
HTTP::Date         -- Модуль анализа дат формата HTTP
HTTP::Negotiate    -- Модуль вычисления градации содержимого HTTP
File::Listing      -- Анализ древовидной структуры директорий сервера
```

Дополнительная документация

Все модули содержат детальную информацию о реализуемых ими интерфейсах. *lwpcook* – это краткое руководство libwww-perl, которое содержит примеры типичного использования модулей библиотеки. Множество полезной информации может дать изучение реализации исходного кода сценариев `lwp-request`, `lwp-rget` и `lwp-mirror`.

[Очень хорошее описание реализации LWP Вы можете найти в книге издательства O'Reilly "Web Client Programming with Perl", изданной в 1997 году. Глава 5 данной книги подробно описывает реализацию libwww-perl, а в главе 6 приведены примеры программ – серверов и клиентов, реализованных при помощи библиотеки LWP. К огромному сожалению, данная книга отсутствует в продаже (даже в США). Со временем, возможно, я переведу основные главы этой книги. Примечание переводчика]

Ошибки

Библиотека до сих пор не поддерживает несколько одновременных запросов. Смотрите также список планируемых доработок в файле TODO.

Благодарности

Данный пакет во многом обязан мотивацией, разработкой и исходным кодом библиотеке libwww-perl для for Perl 4, которую сопровождал Рой Филдинг (fielding@ics.uci.edu).

В пакете LWP использованы фрагменты, написанные Альберто Аккомацци (Alberto Accomazzi), Джеймсом Кейси (James Casey), Брукс Каттер (Brooks Cutter), Мартином Костером (Martijn Koster), Оскаром Нерстражем (Oscar Nierstrasz), Мэлом Мелхнером (Mel Melchner), Гертом ван Остеном (Gertjan van Oosten), Жаредом Рине (Jared Rhine), Ждеком Ширази (Jack Shirazi), Джин Спэффорд (Gene Spafford), Марком Ван Хейнингеном (Marc VanHeyningen), Стивом Бреннером (Steven E. Brenner), Мэрион Хэкенсен (Marion Hakanson), Вальдемаром Кеьбшем (Waldemar Kebsch), Тони Сэндерсом (Tony Sanders), и Лэри Уоллом (Larry Wall); смотрите тексты модулей библиотеки.

Архитекторами – основателями этой библиотеки для Perl 5 являются Мартин Костер (Martijn Koster) и Джисл Аас (Gisle Aas), которым активно помогали Грэм Бар (Graham Barr), Тим Банс (Tim Bunce), Андреас Кёниг (Andreas Koenig), Жаред Рине (Jared Rhine), и Джек Ширази (Jack Shirazi).

Авторские права

Откуда можно загрузить библиотеку

Самая свежая версия данной библиотеки, вероятнее всего, будет доступна на сервере CPAN, а также на: http://www.linpro.no/lwp/

Лучшее место для обсуждения кода библиотеки - список рассылки <libwww@perl.org>.

LWP::SIMPLE – простой процедурный интерфейс LWP

get, head, getprint, getstore, mirror – простой процедурный интерфейс LWP

Синтаксис

```
perl -MLWP::Simple -e 'getprint "http://www.sn.no";'
use LWP::Simple;
$content = get("http://www.sn.no/";)
if (mirror("http://www.sn.no/";, "foo") == RC_NOT_MODIFIED) {
    ...
}
if (is_success(getprint("http://www.sn.no/";))) {
    ...
}
```

Описание

Этот интерфейс предназначен для тех, кто хочет без особых хлопот воспользоваться библиотекой libwww-perl. Он также подойдет и для написания однострочных программ. Если Вам нужно более тонко управлять настройками или получить доступ к полям заголовка в посылаемых запросах и получаемых откликах, Вы должны использовать полностью объектно-ориентированный интерфейс, обеспечиваемый модулем `LWP::UserAgent`.

Модуль обеспечивает реализацию (и экспортирует) следующие функции:

get($url)
: Функция `get()` выбирает документ, идентифицированный данным URL, и возвращает его. Если запрос завершился неудачей, он возвращает `undef`. Аргумент $url может быть либо простой строкой, либо ссылкой на объект URI.

  Вам не удастся проанализировать код возврата или заголовки откликов (типа 'Content-Type') при доступе к Web при помощи данной функции. Если Вы нуждаетесь в этой информации, Вам придется воспользоваться полностью объектно-ориентированным интерфейсом (смотрите описание LWP::UserAgent).

head($url)
: Получает заголовки документа. При успешном завершении возвращает следующие 5 значений ($content_type, $document_length, $modified_time, $expires, $server)
  При неудачном запросе возвращает пустой список. В скалярном контексте возвращает TRUE в случае успешного завершения.

getprint($url)
: Получает и печатает документ, идентифицируемый указанным адресом URL. Документ печатается на STDOUT в качестве данных, полученных из сети. Если запрос завершился неудачно, код завершения и диагностическое сообщение печатается на STDERR. Возвращаемое значение – код отклика HTTP.

getstore($url, $file)
: Получает документ, определенный данным URL и запоминает его в файле. Возвращаемое значение – код отклика HTTP.

mirror($url, $file)
: Получает и сохраняет документ, который определен данным URL, используя *If-modified-since*, и проверяя *Content-Length*. Возвращает код отклика HTTP.

Этот модуль также экспортирует константы и процедуры модуля HTTP::Status. Они могут быть использованы при проверке кода возврата функций `getprint()`, `getstore()` and `mirror()`. Это константы:

```
RC_CONTINUE                      RC_NOT_FOUND
RC_SWITCHING_PROTOCOLS           RC_METHOD_NOT_ALLOWED
RC_OK                            RC_NOT_ACCEPTABLE
RC_CREATED                       RC_PROXY_AUTHENTICATION_REQUIRED
RC_ACCEPTED                      RC_REQUEST_TIMEOUT
RC_NON_AUTHORITATIVE_INFORMATION RC_CONFLICT
RC_NO_CONTENT                    RC_GONE
RC_RESET_CONTENT                 RC_LENGTH_REQUIRED
RC_PARTIAL_CONTENT               RC_PRECONDITION_FAILED
RC_MULTIPLE_CHOICES              RC_REQUEST_ENTITY_TOO_LARGE
RC_MOVED_PERMANENTLY             RC_REQUEST_URI_TOO_LARGE
RC_MOVED_TEMPORARILY             RC_UNSUPPORTED_MEDIA_TYPE
RC_SEE_OTHER                     RC_INTERNAL_SERVER_ERROR
RC_NOT_MODIFIED                  RC_NOT_IMPLEMENTED
RC_USE_PROXY                     RC_BAD_GATEWAY
RC_BAD_REQUEST                   RC_SERVICE_UNAVAILABLE
RC_UNAUTHORIZED                  RC_GATEWAY_TIMEOUT
RC_PAYMENT_REQUIRED              RC_HTTP_VERSION_NOT_SUPPORTED
RC_FORBIDDEN
```

Функции модуля HTTP::Status:

`is_success($rc)`
: Истина, если запрос выполнен успешно

`is_error($rc)`
: Истина, если произошла ошибка при выполнении запроса.

Модуль также экспортирует объект LWP::UserAgent в виде $ua если вы укажете это явным образом.

Пользовательский агент, созданный модулем идентифицирует себя как "LWP::Simple/#.##" (где "#.##" – номер версии libwww-perl) и инициализирует установки прокси-сервера по умолчанию из переменных окружения (путем вызова $ua->env_proxy).

Смотрите также

Описание LWP, Описание LWP::UserAgent, Описание HTTP::Status, *lwp-request*, *lwp-mirror*

LWP::UserAgent - Класс WWW - пользовательский агент (UserAgent)

Синтаксис

```
require LWP::UserAgent;
$ua = LWP::UserAgent->new;
$request = HTTP::Request->new('GET', 'file://localhost/etc/motd');
$response = $ua->request($request); # или
$response = $ua->request($request, '/tmp/sss'); # или
$response = $ua->request($request, \&callback, 4096);
sub callback { my($data, $response, $protocol) = @_; .... }
```

Описание

Класс LWP::UserAgent обеспечивает реализацию простого пользовательского агента WWW для Perl. Он объединяет вместе классы HTTP::Request, HTTP::Response и LWP::Protocol, которые образуют оставшуюся часть ядра библиотеки libwww-perl. В простейших случаях этот класс может быть использован для непосредственного обслуживания запросов WWW, в более сложных –создан потомок для создания приложений с более сложным поведением.

В обычном случае приложение создает объект UserAgent, а затем конфигурирует его, устанавливая значения таймаута, прокси-сервера, имени и т.п. Затем он создает экземпляр объекта HTTP::Request для запроса, который необходимо выполнить. Этот запрос затем передается методу request() класса UserAgent, который отправляет его, применяя соответствующий протокол, и возвращает объект HTTP::Response.

Основной подход библиотеки – использование обмена информации в стиле HTTP для всех типов протоколов, т.е. Вы можете точно так же получать объекты HTTP::Response для запросов по протоколам gopher или ftp. Для достижения полного подобия со стилем HTTP меню gopher и директории ftp-серверов преобразуются в HTML документы.

Метод request() может обработать содержимое отклика тремя возможными способами: в ядре, в файле или путем повторяющихся вызовов подпрограммы. Способ обработки определяется типом второго аргумента, передаваемым функции request().

Вариант обработки в ядре просто запоминает содержимое в скалярном атрибуте 'content' объект – отклика и идеально подходит для небольших откликов HTTP, возможно, нуждающихся в дальнейшем анализе. Этот вариант используется при отсутствии второго аргумента или если второй аргумент не определен (undef).

Вариант с обработкой содержимого в файле требует в качестве второго аргумента request() скаляра, содержащего имя файла и пригоден для больших WWW объектов, которые нужно писать непосредственно в файл, чтобы не затрачивать больших объемов оперативной памяти. В таком случае объект, возвращенный методом request() будет иметь пустой атрибут – контент (содержимое). Если запрос будет неудачен, то содержимое может быть и не пустым, но файл останется нетронутым.

Вариант с подпрограммой требует ссылки на подпрограмму обратного вызова в качестве второго аргумента метода request(), а в качестве необязательного третьего аргумента – размер порции кода. Такой вариант может быть использована для построения "потоковой" обработки, при этом обработка полученных порций может начаться до окончания получения данных. Функция обратного вызова вызывается с тремя аргументами: данными, полученными в данный момент, ссылкой на объект – отклик и ссылкой на объект - протокол. Объект – отклик, возвращаемый методом request(), будет иметь пустое содержимое. Если запрос завершится неудачей, вызывается подпрограмма обратного вызова, а response->content может быть и не пустым.

Запрос может быть отменен при помощи вызова die() в подпрограмме обратного вызова. Сообщение о завершении будет доступно в виде специального поля заголовка "X-Died".

Библиотека также позволяет Вам использовать ссылку на подпрограмму в качестве содержимого объекта – запроса. Будучи вызванной, эта подпрограмма должна возвращать содержимое (возможно, частями), и пустую строку, когда содержимое исчерпано.

Методы

Модулем реализованы следующие методы:

$ua = LWP::UserAgent->new;

Конструктор UserAgent. Возвращает ссылку на объект LWP::UserAgent.

$ua->simple_request($request, [$arg [, $size]])

Этот метод отправляет одиночный WWW запрос от имени пользователя и возвращает полученный ответ. `$request` должен быть ссылкой на объект `HTTP::Request` с определенными, по крайней мере, атрибутами `method()` и `uri()`.

Если `$arg` – скаляр, то его значение используется в качестве имени файла, в котором запоминается содержимое отклика.

Если `$arg` - ссылка на подпрограмму, то она вызывается is a reference to a subroutine, then this routine is called as chunks of the content is received. An optional `$size` argument is taken as a hint for an appropriate chunk size.

Если аргумент `$arg` опущен, то содержимое сохраняется в самом объекте - отклике.

$ua->request($request, $arg [, $size])

Обрабатывает запрос, включая перенаправления и безопасность. Этот метод в действительности может отправить несколько простых различных запросов.

Аргументы аналогичны аргументам метода `simple_request()`.

$ua->redirect_ok

Этот метод вызывается методом `request()` прежде чем он попытается выполнить любое перенаправление. Он должен возвратить значение "ИСТИНА" если разрешено выполнить перенаправление. Производные классы могут аннулировать эту установку.

Реализация по умолчанию возвращает ЛОЖЬ для запросов POST и ИСТИНУ для всех остальных типов.

$ua->credentials($netloc, $realm, $uname, $pass)

Назначает имя пользователя и пароль для использования при доступе к серверу, требующему авторизации ($realm). Часто бывает полезнее вместо данного метода воспользоваться методом `get_basic_credentials()`.

$ua->get_basic_credentials($realm, $uri, [$proxy])

Вызывается методом `request()` для получения удостоверения (обычных параметров) доступа к ресурсу, защищенному обычной или цифровой схемой авторизации доступа.

Должен возвратить имя пользователя и пароль в виде списка. Возвращает undef для запрета попыток определения параметров авторизации.

Данная реализация просто проверяет набор предварительно сохраненных пользовательских переменных. Наследуемые классы могут отменять действие данного метода, например, запрашивая у пользователя имя и пароль. Пример такого использования метода может быть найден в исходном тексте программы `lwp-request`, распространяемой в составе данной библиотеки.

$ua->agent([$product_id])

Получает / присваивает значение имени пользовательского агента для идентификации его в сети. Имя агента отправляется в виде заголовка запроса "User-Agent". По умолчанию используется имя агента "libwww-perl/#.##", где символы "#.##" заменяется номером версии библиотеки.

Строка описания пользовательского агента должна быть совокупностью одного или нескольких идентификаторов продукта, за которым должен следовать номер, отделенный от идентификатора символом слэша “/”. Например:

```
$ua->agent('Checkbot/0.4 ' . $ua->agent);
$ua->agent('Mozilla/5.0');
```

$ua->from([$email_address])

Получает / присваивает значение адреса e-mail лица, ответственного за функционирование пользовательского агента, отправившего запрос. Адрес должен быть пригодным для автоматизированной обработки и соответствовать требованиям спецификации RFC822. Значение отправителя (FROM) отсылается в виде заголовка запроса "From". По умолчанию адрес отсутствует. Пример:

```
$ua->from('gaas@cpan.org');
```

$ua->timeout([$secs])

Получает/устанавливает значение таймаута в секундах. Значение timeout() по умолчанию равно 180 с или 3 минуты.

$ua->cookie_jar([$cookies])

Получает / назначает объект *HTTP::Cookies* для обработки. По умолчанию обработка cookie не предусмотрена, т.е. заголовки "Cookie" автоматически не добавляются к запросам.

$ua->parse_head([$boolean])

Получает/устанавливает индикатор, показывающий, должны ли инициализироваться заголовки из секции <head> документов HTML. Значение по умолчанию – ИСТИНА (TRUE) . Не сбрасывайте это значение, если не понимаете, что может произойти.

$ua->max_size([$bytes])

Получает / устанавливает предельный размер содержимого отклика. По умолчанию установлено значение undef, которое означает отсутствие предела. Если возвращается только часть содержимого отклика, это означает, что превышен предельный установленный размер. В этом случае в отклик добавляется заголовок "X-Content-Range".

$ua->clone;

Возвращает копию объекта LWP::UserAgent

$ua->is_protocol_supported($scheme)

Вы можете воспользоваться данным методом для проверки того, поддерживает ли библиотека libwww-perl указанную схему scheme. Значением $scheme может быть либо обычная строка (типа 'http' или 'ftp'), либо ссылка на объект URI.

$ua->mirror($url, $file)

Получает и записывает в файл документ, на который указывает URL, используя If-Modified-Since, и проверяя Content-Length (создает локальную копию). Возвращает ссылку на объект - отклик.

$ua->proxy(...)

Устанавливает / извлекает адрес URL прокси – сервера для схемы доступа:

```
$ua->proxy(['http', 'ftp'], 'http://proxy.sn.no:8001/');
$ua->proxy('gopher', 'http://proxy.sn.no:8001/');
```

Первая форма определяет, какой адрес URL должен быть использован для проксирования с использованием протоколов доступа, перечисленных в виде списка в качестве первого аргумента, например, 'http' и 'ftp'.

Вторая форма представляет собой сокращенную форму для указания URL прокси-сервера с использованием только одного протокола доступа.

$ua->env_proxy()

Загружает установки прокси-сервера из переменных окружения *_proxy. Вы можете определять параметры прокси, аналогично синтаксису sh:

```
gopher_proxy=http://proxy.my.place/
wais_proxy=http://proxy.my.place/
no_proxy="localhost,my.domain"
export gopher_proxy wais_proxy no_proxy
```

Пользователи оболочек csh и tcsh должны воспользоваться командой setenv для установки этих переменных окружения.

$ua->no_proxy($domain,...)

Не использует прокси-запросы для указанных доменов. Вызов no_proxy без указания параметров очищает список доменов. Например:

```
$ua->no_proxy('localhost', 'no', ...);
```

Смотрите также

Смотрите документацию по модулю LWP для получения полной информации о библиотеке libwww-perl5. В качестве примеров использования смотрите тексты программ *lwp-request* и *lwp-mirror*.

LWP::RobotUA – Класс для реализации Web-роботов

Синтаксис

```
require LWP::RobotUA;
$ua = new LWP::RobotUA 'my-robot/0.1', 'me@foo.com';
$ua->delay(10);  # помедленнее, уважайте посещаемый сервер!
...
# им можно пользоваться, так же как и обычным LWP::UserAgent
$res = $ua->request($req);
```

Описание

Данный класс реализует пользовательский агент, пригодный для реализации приложений – роботов. Роботы должны относиться с уважением к серверам, которые они посещают. Они должны вначале ознакомиться с правилами, установленными для роботов в файле */robots.txt,* для того, чтобы убедиться, что посещения роботов приветствуются; Кроме того, они не должны посылать запросы посещаемым серверам слишком часто.

Но, прежде чем Вы начнете писать свое приложение – робота, ознакомьтесь с документом http://info.webcrawler.com/mak/projects/robots/robots.html.

Если Вы используете *LWP::RobotUA* в качестве своего пользовательского агента, Вам не нужно ломать себе голову о том, как соблюсти все выдвигаемые условия самостоятельно. Просто посылайте запросы, как если бы Вы пользовались обычным пользовательским агентом *LWP::UserAgent,* а этот специальный агент *LWP::RobotUA* сделает все, как положено.

Методы

LWP::RobotUA является дочерним классом класса LWP::UserAgent и реализует те же самые методы. Кроме того, он дополнительно обеспечивает реализацию следующих методов:

$ua = LWP::RobotUA->new($agent_name, $from, [$rules])

Имя Вашего робота, а также e-mail адрес лица, ответственного за функционирование робота (т.е. Вас) являются обязательными параметрами конструктора.

Вы можете, по желанию, указать в качестве параметра имя используемого объекта *WWW::RobotRules* – правил поведения робота.

$ua->delay([$minutes])

Устанавливает минимальную задержку между запросами к одному и тому же серверу. По умолчанию время задержки равно 1 минуте.

$ua->use_sleep([$boolean])
Получает / устанавливает значение, показывающее, должен ли пользовательский агент "заснуть" (sleep()) если запрос приходит слишком быстро (прежде чем пройдет $ua->delay минут). Значение по умолчанию - TRUE. Если значение – FALSE то будет сгенерирован внутренний отклик SERVICE_UNAVAILABLE. Он будет иметь заголовок с полем Retry-After, который показывает, когда будет разрешено отправить еще один запрос данному серверу.

$ua->rules([$rules])
Присваивает/извлекает информацию об используемом объекте *WWW::RobotRules*.

$ua->no_visits($netloc)
Возвращает количество документов, принятых с данного сервера. Конечно же, этот метод должен называться num_visits() или подобным образом, но… ☹

$ua->host_wait($netloc)
Возвращает время ожидания в секундах до посылки следующего запроса к данному хосту (начиная с текущего момента).

$ua->as_string
Возвращает строку, описывающую состояние пользовательского агента. Полезен, главным образом, для целей отладки.

Смотрите также:

LWP::UserAgent, WWW::RobotRules

WWW::RobotsRules анализатор файлов robots.txt

Синтаксис

```
require WWW::RobotRules;
my $robotsrules = new WWW::RobotRules 'MOMspider/1.0';
use LWP::Simple qw(get);
$url = "http://some.place/robots.txt";;
my $robots_txt = get $url;
$robotsrules->parse($url, $robots_txt);
$url = "http://some.other.place/robots.txt";;
my $robots_txt = get $url;
$robotsrules->parse($url, $robots_txt);
# Сейчас мы можем проверить допустимость сканирования URL на основе правил,
# которые получены при анализе файла "robots.txt" на данном сервере.
if($robotsrules->allowed($url)) {
    $c = get $url;
    ...
}
```

Описание

Этот модуль осуществляет синтаксический анализ файла */robots.txt* как определено в “Стандарте правил запрета для роботов (A Standard for Robot Exclusion)”, описанном в документе http://info.webcrawler.com/mak/projects/robots/norobots.html.

Вебмастера могут использовать файл */robots.txt* для запрета доступа к некоторым частям их сайтов роботов, соответствующих вышеуказанному стандарту.

Анализируемый файл сохраняется в объекте WWW::RobotRules; этот объект обеспечивает методы проверки запрета доступа к данному URL. Один объект WWW::RobotRules может анализировать множество файлов */robots.txt*.

Реализованы следующие методы:

$rules = WWW::RobotRules->new($robot_name)
: Это конструктор объектов WWW::RobotRules. Аргумент, передаваемый new() – это имя робота.

$rules->parse($robot_txt_url, $content, $fresh_until)
: Метод parse() принимает в качестве аргументов URL, который был использован для получения файла /robots.txt, а также содержимое этого файла.

$rules->allowed($uri)
: Возвращает значение ИСТИНА, если робот позволяет загрузить документ с данным URL.

$rules->agent([$name])
: Получить/установить имя агента. ПРИМЕЧАНИЕ: Изменение имени агента очищает кэшированные значения правил robots.txt, а также срок действия правил.

ROBOTS.TXT

Формат и семантика файла /robots.txt следующая (приведем отредактированный краткий реферат документа http://info.webcrawler.com/mak/projects/robots/norobots.html):

Файл состоит из одной или более записей, разделенных между собой одной или несколькими пустыми строками. Каждая запись содержит строки вида:

```
<имя-поля>: <значение>
```

Имя поля нечувствительно к регистру символов. Текст после символа '#' до конца строки игнорируется и используется в качестве комментария. Могут быть использованы следующие <имена - полей>:

User-Agent
: Значением этого поля будет имя робота, для которого описывается политика доступа. Если имеется более одного поля *User-Agent,* запись описывает идентичную политику доступа для более чем одного робота. В записи должно присутствовать, по крайней мере, одно поле. Если значением поля является '*', запись описывает политику доступа по умолчанию для любого робота, не соответствующему условиям других записей.

Disallow
: Значение этого поля определяет частичный URL, который запрещено посещать роботу. Это может быть как частичным, так и полным путем; любой URL, который начинается с этого значения, не будет обработан роботом

Примеры файлов ROBOTS.TXT

Данный пример файла "/robots.txt'' определяет, что всем роботам запрещено посещать документы с адресом URL, начинающимся с /cyberworld/map/ или /tmp/:

```
User-agent: *
Disallow: /cyberworld/map/ # за этими воротами – виртуальная бесконечность…
Disallow: /tmp/ # а эти файлы скоро исчезнут
```

Файл /robots.txt из нижеследующего примера указывает, что роботам запрещено посещать любые документы, чей адрес URL начинается с /cyberworld/map/, за исключением робота с именем "cybermapper":

```
User-agent: *
```

```
Disallow: /cyberworld/map/ # за этими воротами – виртуальная бесконечность…
# А робот Cybermapper может все!
User-agent: cybermapper
Disallow:
```

Следующий пример запрещает в дальнейшем всем роботам посещать данный сайт:

```
# А ну-ка, пошли все прочь!
User-agent: *
Disallow: /
```

Смотрите также

LWP::RobotUA, WWW::RobotRules::AnyDBM_File

WWW::RobotRules::AnyDBM_File постоянно хранимые правила обработки файлов /robots.txt - RobotRules

Синтаксис

```
require WWW::RobotRules::AnyDBM_File;
require LWP::RobotUA;
# Создает пользовательский агент-робот, который использует  RobotRules,
# сохраняемый в кэше на диске
my $rules = new WWW::RobotRules::AnyDBM_File 'my-robot/1.0', 'cachefile';
my $ua = new WWW::RobotUA 'my-robot/1.0', 'me@foo.com', $rules;
# Затем используем пользовательский агент $ua как обычно
$res = $ua->request($req);
```

Описание

Это дочерний класс класса *WWW::RobotRules*, который использует пакет AnyDBM_File для постоянного хранения в дисковом кэше правил, установленных файлом *robots.txt,* а также информации о посещении данного хоста.

Конструктор (метод new()) использует дополнительный аргумент, который определяет имя используемого файла DBM. Если файл DBM уже существует, Вы можете указать в качестве имени агента undef, так как это имя может быть взято из базы данных DBM.

Смотрите также

WWW::RobotRules, LWP::RobotUA

LWP::Debug - подпрограммы отладки для библиотеки libwww-perl

Синтаксис

```
use LWP::Debug qw(+ -conns);
# Используется внутри библиотеки LWP
LWP::Debug::trace('send()');
```

```
LWP::Debug::debug('url ok');
LWP::Debug::conns("Прочитано $n байт: $data");
```

Описание

LWP::Debug предоставляет возможность трассировки. Функции trace(), debug() and conns() вызываются внутри библиотеки и регистрируют информацию с повышенным уровнем детализации. Реальный уровень детализации вывода диагностики управляются при помощи функции level().

Доступны следующие функции:

level(...)
Функция level() управляет уровнем детализации фиксируемых событий. Передача '+' или '-' указывает на максимальный уровень детализации или отсутствие вывода диагностических сообщений соответственно. Можно включить или отключить индивидуальные уровни детализации, передав уровня имя функции level(), предварив его знаком '+' или '-'. Доступны следующие уровни детализации вывода диагностики:

```
  trace    : трассировать вызовы функций
  debug    : Печатать отладочные сообщения
  conns    : Отображает все данные передаваемые через соединения
```

Модуль LWP::Debug реализует специальный метод import() который позволяет Вам передать аргументы level() в начальном предложении use. Если аргумент начинается со знака '+' или '-' то он передается функции level(), иначе имя экспортируется как обычно. Следовательно, следующие два предложения являются эквивалентными (если Вы игнорируете тот факт, что второй пример захламляет Ваше пространство имен):

```
  use LWP::Debug qw(+);
  use LWP::Debug qw(level); level('+');
```

trace($msg)
Функция trace() используется для трассировки вызовов функций пакета. Печатается наименование пакета и имя вызывающей подпрограммы, а также переданные параметры. Она должна быть вызвана при запуске любой из основных функций пакета.

debug($msg)
Функция debug() используется в функциях для получения подробного отчета о состоянии.

conns($msg)
Функция conns() используется для отображения данных, передаваемых через соединение. Она может сгенерировать достойный внимания отчет.

LWP::MediaTypes предполагаемый тип содержимого файла или URL

Синтаксис

```
use LWP::MediaTypes qw(guess_media_type);
$type = guess_media_type("/tmp/foo.gif");
```

Описание

Данный модуль реализует функции для оперирования различными типами содержимого - контента (иными словами, типами MIME). Файл *media.types* назначает соответствие типов контента расширениям файлов. Если существует файл *~/.media.types*, то для соответствия используется именно он. Для обратной совместимости с предыдущими версиями модуль также ищет файл *~/.mime.types*.

По умолчанию экспортируются следующие функции:

guess_media_type($filename_or_url, [$header_to_modify])

Данная функция пытается предположить тип содержимого и кодировку файла или адреса url. Она возвращает тип содержимого, который представляет собой строку вида "text/html". В контексте массива она также возвращает примененные кодировки содержимого (в порядке, использованном для кодирования файла). Функции можно также передать ссылку на объект URI вместо имени файла.

Если вывод о типе содержимого не может быть сделан, исходя из имени файла, то функция guess_media_type()передаст право определения типа содержимого оператору Perl -T. Если это работает (и -T возвращает истинное значение – TRUE), то функция возвратит в качестве типа содержимого *text/plain,* в противном случае она возвратит *application/octet-stream.*

Необязательный второй аргумент должен быть ссылкой на объект HTTP::Headers, либо на любой другой объект, который реализует подобным образом метод $obj->header. При наличии второго аргумента для данного заголовка будут установлены значения 'Content-Type' и 'Content-Encoding'.

media_suffix($type,...)

Данная функция возвращает все расширения файлов, которые используются для обозначения указанного типа (ов) содержимого. В обозначениях типов могут использоваться групповые символы (* и т.д.). В скалярном контексте возвращает первый найденный тип файла.

Примеры:

```
@suffixes = media_suffix('image/*', 'audio/basic');
$suffix = media_suffix('text/html');
```

Следующие функции экспортируются по явному запросу:

add_type($type, @exts)

Связывает список расширений файлов с данным типом содержимого.
Пример:

```
add_type("x-world/x-vrml" => qw(wrl vrml));
```

add_encoding($type, @ext)

Связывает список расширений файлов с типом кодирования данных.

Пример:

```
add_encoding("x-gzip" => "gz");
```

read_media_types(@files)

Анализирует файлы типов информации и добавляет в список обрабатываемых типов найденные в них соответствия.

Пример:

```
read_media_types("conf/mime.types");
```

LWP::Protocol
Базовый класс для протоколов LWP

Синтаксис

```
package LWP::Protocol::foo;
require LWP::Protocol;
@ISA=qw(LWP::Protocol);
```

Описание

Этот класс используется в качестве базового класса для всех реализаций протоколов, поддерживаемых библиотекой LWP.

При создании экземпляра данного класса при помощи LWP::Protocol::create($url) Вы получаете инициализированный производный класс, соответствующий указанному методу доступа. Другими словами, функция LWP::Protocol::create() вызывает конструктор одного из этих производных классов.

Все наследуемые классы класса LWP::Protocol должны подменить метод request(), который используется для обслуживания запроса. Подмененный метод может воспользоваться функцией collect() для сбора порций данных по мере их получения.

Данным классом реализованы следующие методы и функции:

$prot = LWP::Protocol->new()
: Конструктор LWP::Protocol наследуется производными классами. Так как это виртуальный базовый класс, этот метод **НЕ ДОЛЖЕН** вызываться непосредственно.

$prot = LWP::Protocol::create($url)
: Создает объект класса, включая протокол (схему), необходимый для доступа к данному URL. Это - функция, а не метод. Это, – скорее, генератор объектов, а не конструктор. Это функция, которую обязан использовать пользовательский агент для доступа к протоколам.

$class = LWP::Protocol::implementor($scheme, [$class])
: Получает и/или инициализирует класс-конструктор для протокола (схемы). Возвращает "", если указанная схема не поддерживается.

$prot->request(...)

```
$response = $protocol->request($request, $proxy, undef);
$response = $protocol->request($request, $proxy, '/tmp/sss');
$response = $protocol->request($request, $proxy, \&callback, 1024);
```

: Передает запрос с использованием соответствующего протокола и возвращает объект – отклик. Этот метод должен быть заменен соответствующим методом производного класса. Смотрите описание аргументов в руководстве LWP::UserAgent.

$prot->timeout($seconds)
: Получает или устанавливает значение таймаута в секундах.

$prot->parse_head($yesno)
: Нужно ли инициализировать заголовки откликов из разделов <head> документов HTML.

$prot->collect($arg, $response, $collector)
: Вызывается для того, чтобы собрать воедино содержимое отклика на запрос и обработать его, сохранив его в скаляре, файле или при помощи функции обратного вызова. Если аргумент $arg не определен, то содержимое записывается в $response. Если $arg представляет собой простой скаляр, то $arg интерпретируется в качестве имени файла и полученное содержимое отклика пишется в этот файл. Если $arg

представляет собой ссылку на подпрограмму, то содержимое отклика передается для обработки этой подпрограмме.

$collector – это вызываемая подпрограмма, отвечающая за перенаправление частей содержимого для обработки (в качестве ссылок на скаляр). Подпрограмма $collector возвращает пустую строку в качестве сигнала достижения конца файла (EOF).

Значение, возвращаемое collect() – это ссылка на объект $response.

Примечание: Аргумент – файл или процедура обратного вызова будет использована только в случае истинности $response->is_success(). Это предотвращает отправку содержимого для откликов перенаправления и проверки доступа, что может сбить с толку процедуру обратного вызова.

$prot->collect_once($arg, $response, $content)

Может быть вызван, когда в качестве $content получен весь отклик целиком. Этот метод вызовет collect() с процедурой – коллектором обратного вызова в качестве параметра, которая возвратит ссылку на $content – содержимое при первом вызове и пустую строку при следующем..

Смотрите также

Внимательно изучите примеры использования класса в файлах *LWP/Protocol/file.pm* и *LWP/Protocol/http.pm*.

LWP::MemberMixin
класс для доступа к переменным библиотеки LWP

Синтаксис

```
package Foo;
require LWP::MemberMixin;
@ISA=qw(LWP::MemberMixin);
```

Описание

Класс mixin обеспечивает метод, обеспечивающий непосредственный доступ к переменным - членам класса в %$self. В идеале использование данного метода является лучшим решением с точки зрения Perl.

Класс обеспечивает реализацию только одного метода:

_elem($elem [, $val])

Внутренний метод для получения / установки значения переменной `$elem` – члена класса пакета LWP. Если значение параметра `$val` определено, то оно используется в качестве нового значения переменной, если нет - значение переменной не изменяется. В обоих случаях возвращается предыдущее значение переменной- члена класса.

HTTP::Headers
Класс, инкапсулирующий заголовки сообщений HTTP

Синтаксис

```
require HTTP::Headers;
$h = new HTTP::Headers;
```

Описание

Класс HTTP::Headers инкапсулирует заголовки сообщений HTTP-типа. Заголовки состоят из пар атрибут-значение, которые могут повторяться и которые выводятся в выходной поток в определенном порядке.

Экземпляры данного класса обычно создаются в качестве внутренних переменных классов HTTP::Request и HTTP::Response, и являются внутренними по отношению к библиотеке.

Реализованы следующие методы:

$h = new HTTP::Headers

Порождает новый экземпляр объекта HTTP::Headers. Вы можете передать конструктору некоторые исходные пары атрибут-значение. *Например*:

```
$h = new HTTP::Headers
    Date         => 'Thu, 03 Feb 1994 00:00:00 GMT',
    Content_Type => 'text/html; version=3.2',
    Content_Base => 'http://www.sn.no/';
```

$h->header($field [=> $value],...)

Получает или устанавливает значение заголовка. Имя поля заголовка не чувствительно к регистру символов. Для того, чтобы облегчить жизнь тем пользователям perl, кто хочет избежать использования скобок перед оператором =>, допускается использование символа подчеркивания '_' в качестве синонима для '-' в именах полей (такое поведение может быть запрещено путем присвоения значения FALSE переменной $HTTP::Headers::TRANSLATE_UNDERSCORE).

Методу header()можно передать в качестве параметров множество пар ($field => $value); таким образом, Вы можете обновить несколько полей одновременно, вызвав метод только однажды.

Необязательный аргумент $value может быть либо скаляром, либо ссылкой на список скаляров. Если аргумент $value не определен или не задан, то заголовок не изменяется.

Метод возвращает предыдущие значения $field. Поля, содержащие не единственное значение, в скалярном контексте будут сцеплены при помощи символа ",".

```
$header->header(MIME_Version => '1.0',
                User_Agent   => 'My-Web-Client/0.01');
$header->header(Accept => "text/html, text/plain, image/*");
$header->header(Accept => [qw(text/html text/plain image/*)]);
@accepts = $header->header('Accept');
```

$h->scan(\&doit)

Применяет подпрограмму к каждому заголовку по очереди. Процедура обратного вызова вызывается с двумя параметрами: именем поля и одиночным значением. Если заголовок имеет более одного значения, подпрограмма вызывается для каждого значения. Имена полей, переданных подпрограмме, имеют регистр символов, который

предлагается спецификацией HTTP, а заголовки будут обрабатываться в порядке, предписанном правилами "хорошего тона" (Good Practice).

$h->as_string([$endl])

Возвращает поля заголовка в виде форматированных MIME – заголовков. В связи с тем, что данный метод использует внутренний вызов метода scan() для создания результирующей строки, результат будет полностью соответствовать предписаниям спецификации HTTP, а также следовать "правилам хорошего тона" при определении порядка обработки полей. Длинные значения заголовков не переносятся на следующую строку.

Необязательный параметр определяет последовательность символов, используемую для обозначения конца строки. По умолчанию концом строки считается "\n". Символы "\n", внедренные в заголовок, будут заменены такой концевой последовательностью.

$h->push_header($field, $val)

Добавляет новое поле к указанному заголовку. Имя поля заголовка не чувствительно к регистру символов. Поле не обязательно должно иметь ранее установленное значение. Предыдущие значения для одинаковых полей сохраняются. Аргумент может быть либо скаляром, либо ссылкой на список скаляров.

```
 $header->push_header(Accept => 'image/jpeg');
```

$h->remove_header($field,...)

Данная функция удаляет заголовки с указанными именами.

$h->clone

Возвращает копию данного объекта HTTP::Headers.

Методы, обеспечивающие удобный доступ к заголовкам

Можно получить доступ к часто используемым заголовкам при помощи описанных ниже методов, обеспечивающих удобный доступ. Эти методы могут быть использованы как для чтения, так и для присвоения значения заголовкам. Значение присваивается заголовку, при передаче аргумента методу. Метод всегда возвращает предыдущее значение заголовка.

Методы, работающие с датой/временем, всегда преобразовывают свое значение в системное время (количество секунд, прошедших с 1 января 1970 года), а также ожидают значения в таком же формате в качестве параметра, при присвоении значения заголовку.

$h->date

Это заголовок представляет собой дату и время создания сообщения. Например:

```
  $h->date(time);  # установить текущее время
```

$h->expires

Данный заголовок указывает дату и время окончания срока жизни объекта.

$h->if_modified_since

$h->if_unmodified_since

Данный заголовок используется для того, чтобы ввести проверку условия в запрос. Если запрашиваемый ресурс был (не был) изменен со времени, указанного в данном поле, то сервер возвратит отклик "304 Not Modified" (документ не изменялся) вместо самого документа.

$h->last_modified

Данный заголовок показывает дату и время последней модификации ресурса. Например:

```
  # проверить прошло ли больше часа с момента модификации документа
  if ($h->last_modified < time - 60*60) {
        ...
  }
```

$h->content_type

Поле заголовка Content-Type показывает тип содержимого, передаваемого сообщения. Например:

```
  $h->content_type('text/html');
```

Возвращаемое значение будет преобразовано в нижний регистр, а параметры будут очищены от символов конца строки и возвращены по отдельности при вызове метода в скалярном контексте. Это делает возможным, к примеру, без опасений выполнить следующий фрагмент кода:

```
if ($h->content_type eq 'text/html') {
 # Мы можем записать проверку даже так, несмотря на то, что реальный заголовок
 # будет таким: 'TEXT/HTML; version=3.0'
  ...
}
```

$h->content_encoding

Поле заголовка Content-Encoding используется в качестве модификатора указанного типа содержимого. При наличии данного метода, его значение показывает, какой дополнительный механизм кодирования был применен к источнику.

$h->content_length

Десятичное число – размер в байтах тела сообщения.

$h->content_language

Язык(и) которые используются в сообщении. Значением являются один или более тэгов – описателей языка, определенных спецификацией RFC 1766, например,"no" для норвежского (Norvegian) и "en-US" для американского английского (US-English).

$h->title

Заголовок документа. В библиотеке libwww-perl данный заголовок будет автоматически инициализирован содержимым тэга <TITLE>...</TITLE> документа HTML. *Данный заголовок больше не является частью текущего стандарта HTTP.*

$h->user_agent

Данное поле заголовка используется в сообщениях – запросах и содержит информацию о пользовательском агенте, отправившем данный запрос. Например:

```
$h->user_agent('Mozilla/1.2');
```

$h->server

Поле заголовка "server" содержит информацию о программном обеспечении сервера, отправившего запрос.

$h->from

Данный заголовок должен содержать адрес e-mail лица, ответственного за управление пользовательским агентом, отправившим запрос. Адрес должен быть пригодным для автоматизированной обработки как определено спецификацией RFC822. Например.:

```
$h->from('Gisle Aas <aas@sn.no>');
```

$h->referer

Используется для указания адреса (URI) документа, с которого был получен запрашиваемый адрес ресурса.

$h->www_authenticate

Данный заголовок должен быть включен в качестве неотъемлемой части отклика "401 Unauthorized" (Доступ запрещен). Значение поля состоит из запроса, который указывает способ доступа и принимаемые параметры для запрашиваемого URI.

$h->proxy_authenticate

Данный заголовок должен быть включен в отклик "407 Proxy Authentication Required" (требуется авторизация пользователя для доступа к прокси-серверу).

$h->authorization

$h->proxy_authorization

Пользовательский агент, которому нужно получить доступ к сереверу или к прокси – серверу, получает возможность регистрации, включив такие заголовки.

$h->authorization_basic

Данный метод используется для получения или присвоения значения для авторизационного заголовка, который использует базовую схему контроля доступа. Будучи вызванным в контексте массива он возвратит два значения – имя пользователя и пароль. В скалярном контексте он возвратит строку в формате "*uname:password*".

При присвоении значения заголовку, метод ожидает два аргумента. Например:

```
$h->authorization_basic($uname, $password);
```

Метод выдаст сообщение об ошибке, если $uname содержит двоеточие ':'.

$h->proxy_authorization_basic

Аналогичен методу authorization_basic(), за исключением того, что он присваивает значение заголовку "Proxy-Authorization".

HTTP::Headers::Util
Служебные функции для анализа заголовков

Синтаксис

```
use HTTP::Headers::Util qw(split_header_words);
@values = split_header_words($h->header("Content-Type"));
```

Описание

Данный модуль обеспечивает реализацию нескольких функций, которые помогают осуществить синтаксический анализ и присвоить правильные значения полям заголовков HTTP. По умолчанию ни одна из этих функций не экспортируется.

Реализованы следующие функции:

split_header_words(@header_values)

Данная функция разбирает значения полей заголовка, переданные ей в качестве аргумента, в список анонимных массивов, содержащих пары ключ/значение. Функция знает, как можно правильно обработать символы "," , ";" и "=", так же как и значения, заключенные в кавычки, следующие за символом присваивания "=". Список лексем, разделенных пробелами, обрабатывается так же, как если бы лексемы были разделены символом ";".

Если @header_values переданные в качестве аргумента, состоят из нескольких значений, то они трактуются, как если бы они были одиночными значениями, разделенными запятыми ",".

Это означает, что данная функция полезна для анализа полей заголовка, которые следуют данному синтаксису (в виде БНФ – формы Бэкуса-Наура, в качестве формы спецификации HTTP/1.1, но ослабим требования для лексем).

```
заголовки          = #заголовок
заголовок          = (лексема | параметр) *( [";"] (лексема | параметр))
лексема            = 1*<любой СИМВОЛ за исключением УПРАВЛЯЮЩИХ или
                        разделителей>
разделители        = "(" | ")" | "<" | ">" | "@"
                   | "," | ";" | ":" | "\" | <">
                   | "/" | "[" | "]" | "?" | "="
                   | "{" | "}" | SP | HT
строка в кавычках  = ( <"> *(qdtext | quoted-pair ) <"> )
qdtext             = <любой текст за исключением символа <">>
quoted-pair        = "\" СИМВОЛ
параметр           = атрибут "=" значение
атрибут            = лексема
значение           = лексема | строка в кавычках
```

Каждый заголовок *header* представляется анонимным массивом, состоящим из пар ключ/значение. Значением простой лексемы (не части параметра) является undef. Синтаксически некорректные заголовки по Вашему желанию могут не анализироваться.

Проще всего это можно показать на конкретных примерах. Утверждения:

```
split_header_words('foo="bar"; port="80,81"; discard, bar=baz')
split_header_words('text/html; charset="iso-8859-1");
split_header_words('Basic realm="\"foo\\bar\""');
```

возвратят

```
[foo=>'bar', port=>'80,81', discard=> undef], [bar=>'baz' ]
['text/html' => undef, charset => 'iso-8859-1']
[Basic => undef, realm => '"foo\bar"']
```

join_header_words(@arrays)

Выполняет преобразование, противоположное по действию преобразованию, выполняемому функцией split_header_words(). Она получает в качестве параметров список анонимных массивов (или список пар ключ/значение) и создает цельный заголовок. При необходимости значения атрибутов заключаются в кавычки.

Пример:

```
join_header_words(["text/plain" => undef, charset => "iso-8859/1"]);
join_header_words(""text/plain" => undef, charset => "iso-8859/1"");
```

оба утверждения возвратят значение заголовка:

```
text/plain; charset="iso-8859/1"
```

HTTP::Message
Класс, инкапсулирующий сообщения HTTP

Синтаксис

```
package HTTP::Request;  # or HTTP::Response
require HTTP::Message;
@ISA=qw(HTTP::Message);
```

Описание

Объект HTTP::Message содержит некоторые заголовки и содержимое (тело). Данный класс является абстрактным, т.е. используется только в качестве базового класса для HTTP::Request и HTTP::Response и никогда не обрабатывается сам.

Доступны следующие методы:

$mess = new HTTP::Message;

Конструктор объекта. Должен вызываться только внутренними процедурами библиотеки. Внешний код должен оперировать только объектами HTTP::Request или HTTP::Response.

$mess->clone()

Возвращает копию объекта.

$mess->protocol([$proto])

Устанавливает протокол HTTP, используемый для обработки сообщения. protocol() представляет собой строку в виде "HTTP/1.0" или "HTTP/1.1".

$mess->content([$content])

Метод content() устанавливает значение содержимого (контента), если ему передается аргумент. Если аргумент не указан, содержимое не изменяется. В любом случае метод возвращает предыдущее содержимое.

$mess->add_content($data)
Метод add_content() дописывает дополнительные данные в конец существующего контента.

$mess->content_ref
Метод content_ref() возвращает ссылку на строку содержимого (контента). Такой способ может быть более эффективным для доступа к содержимому большого размера, а также может использоваться для непосредственной обработки содержимого.
Например:

```
${$res->content_ref} =~ s/\bfoo\b/bar/g;
```

$mess->headers;
Возвращает встроенный объект HTTP::Headers.

$mess->headers_as_string([$endl])
Вызывает метод HTTP::Headers->as_string()для обработки заголовков сообщения.

Неизвестные методы HTTP::Message делегируются объекту HTTP::Headers, который является частью любого сообщения. Это позволяет получить удобный доступ к таким методам. Обратитесь к руководству HTTP::Headers для получения детальной информации о методах, перечисленных ниже:

```
$mess->header($field => $val);
$mess->scan(\&doit);
$mess->push_header($field => $val);
$mess->remove_header($field);
$mess->date;
$mess->expires;
$mess->if_modified_since;
$mess->if_unmodified_since;
$mess->last_modified;
$mess->content_type;
$mess->content_encoding;
$mess->content_length;
$mess->content_language
$mess->title;
$mess->user_agent;
$mess->server;
$mess->from;
$mess->referer;
$mess->www_authenticate;
$mess->authorization;
$mess->proxy_authorization;
$mess->authorization_basic;
$mess->proxy_authorization_basic;
```

HTTP::Request
Класс, инкапсулирующий запросы HTTP

Синтаксис

```
require HTTP::Request;
$request = HTTP::Request->new(GET => 'http://www.oslonett.no/');
```

Описание

HTTP::Request – это класс, инкапсулирующий запросы в стиле HTTP style, состоящие из строки запроса, нескольких заголовков и некоторого (возможно, отсутствующего) содержимого – контента. Обратите внимание на то, что библиотека LWP также использует такой стиль запросов и для обслуживания протоколов, отличных от HTTP.

Экземпляры этого класса обычно передаются методу request() объекта LWP::UserAgent:

```
$ua = LWP::UserAgent->new;
$request = HTTP::Request->new(GET => 'http://www.oslonett.no/');
$response = $ua->request($request);
```

HTTP::Request является дочерним классом HTTP::Message и, следовательно, наследует его методы. Наиболее часто используемые наследуемые методы: header(), push_header(), remove_header(), headers_as_string() и content(). Полное описание можно найти в руководстве HTTP::Message.

Реализованы следующие дополнительные методы:

$r = HTTP::Request->new($method, $uri, [$header, [$content]])
: Создает новый объект HTTP::Request описывающий запрос к объекту $uri с использованием метода $method. Аргумент $uri может быть либо строкой, либо ссылкой на объект URI. Аргумент $header должен быть ссылкой на объект HTTP::Headers.

$r->method([$val])
$r->uri([$val])
: Эти методы обеспечивают общий доступ к атрибутам, содержащим, соответственно, метод запроса и URI объекта, которому адресуется запрос.
Если указан аргумент, то он становится новым значением. Если аргумент не указан, значение не изменяется. В обоих случаях методы возвращают предыдущее значение.

: Методу uri() можно передать в качестве аргумента как ссылку на объект URI, так и обычную строку. В случае, если передается строка, она должна быть распознана в качестве абсолютного URI.

$r->as_string()
: Метод, возвращающий текстовое представление запроса. Полезен, в основном, для целей отладки. Не имеет аргументов.

Смотрите также

HTTP::Headers, HTTP::Message, HTTP::Request::Common

HTTP::Request::Common
создание обычных объектов HTTP::Request

Синтаксис

```
use HTTP::Request::Common;
$ua = LWP::UserAgent->new;
$ua->request(GET 'http://www.sn.no/');
$ua->request(POST 'http://somewhere/foo', [foo => bar, bar => foo]);
```

Описание

Данный модуль обеспечивает реализацию функций, которые возвращают вновь созданные объекты HTTP::Request. Эти функции обычно более удобны при создании обычных запросов, чем стандартный конструктор HTTP::Request. Модулем реализованы следующие функции.

GET $url, Header => Value,...

Функция GET() возвращает объект HTTP::Request, который инициализирован с параметрами - методом GET и указанным адресом URL. Если дополнительные аргументы не указаны, то он в точности эквивалентен следующему фрагменту:

```
HTTP::Request->new(GET => $url)
```

но гораздо менее громоздок. Он также лучше читается при совместном использовании с методом LWP::UserAgent->request():

```
my $ua = new LWP::UserAgent;
my $res = $ua->request(GET 'http://www.sn.no')
if ($res->is_success) { ...
```

Вы можете также инициализировать значения в заголовках запроса путем указания пар ключевое слово/значение в качестве дополнительных аргументов. Например:

```
$ua->request(GET 'http://www.sn.no',
                 If_Match => 'foo',
                 From     => 'gisle@aas.no',
            );
```

Ключевое слово 'Content' является зарезервированным и, если оно используется в качестве параметра метода, то оно инициализирует содержательную часть запроса (контент), вместо создания заголовка.

HEAD $url, [Header => Value,...]

Подобен методу GET(), но в качестве метода запроса используется HEAD.

PUT $url, [Header => Value,...]

Подобен методу GET(), но в качестве метода запроса используется PUT.

POST $url, [$form_ref], [Header => Value,...]

Метод, в основном, работает подобно методу GET(), используя POST в качестве метода запроса, но данная функция может иметь и второй (необязательный) аргумент – массив или ссылку на хэш ($form_ref). Этот аргумент может использоваться для передачи форме пар ключевое слово/значение. По умолчанию запрос инициализируется с типом содержимого - content-type application/x-www-form-urlencoded. Это означает, что Вы можете эмулировать реализацию метода POST в формах HTML , например, так:

```
POST 'http://www.perl.org/survey.cgi',
     [ name   => 'Gisle Aas',
       email  => 'gisle@aas.no',
       gender => 'M',
       born   => '1964',
       perc   => '3%',
     ];
```

Данная конструкция создаст новый объект HTTP::Request , который будет вот таким:

```
POST http://www.perl.org/survey.cgi
Content-Length: 66
Content-Type: application/x-www-form-urlencoded
name=Gisle%20Aas&email=gisle%40aas.no&gender=M&born=1964&perc=3%25
```

(Примечание переводчика: Конечно, можно реализовать подобный запрос и с помощью модуля CGI.pm, но для самостоятельно действующих web-программ подход LWP оказывается более предпочтительным).

Метод POST также поддерживает кодировку содержимого формы multipart/form-data, используемую для выгрузки файлов на удаленный сервер, как определено спецификацией RFC1867. Вы можете переключаться между кодировками формы, путем указания типа содержимого 'form-data' в качестве одного из заголовков запроса. Если одно из значений $form_ref является ссылкой на массив, то оно рассматривается в качестве смешанного описателя файла и интерпретируется следующим образом:

```
[ $file, $filename, Header => Value... ]
```

Первый элемент массива ($file) – имя открываемого файла. Этот файл будет прочитан, и его содержимое будет помещено в запрос. Подпрограмма умрет, если файл не может быть открыт. Для того, чтобы определить содержимое непосредственно, используйте undef в качестве значения переменной $file. $filename - это filename для отчета о результате запроса. Если этот значение не определено, то будет использовано базовое имя файла $file. Вы можете указать в качестве $filename пустую строку, если не хотите указывать какое-либо имя файла в запросе.

Для того, чтобы вынести файл *~/.profile* на всеобщее обозрение, которое было продемонстрировано выше (survey), можно воспользоваться таким фрагментом кода:

```
POST 'http://www.perl.org/survey.cgi',
     Content_Type => 'form-data',
     Content      => [ name  => 'Gisle Aas',
                       email => 'gisle@aas.no',
                       gender => 'M',
                       born   => '1964',
                       init   => ["$ENV{HOME}/.profile"],
                     ]
```

Будет создан объект HTTP::Request, который будет выглядеть, примерно, так (ограничитель секций и содержимое Вашего файла, вероятно, будут отличаться):

```
POST http://www.perl.org/survey.cgi
Content-Length: 388
Content-Type: multipart/form-data; boundary="6G+f"
--6G+f
Content-Disposition: form-data; name="name"

Gisle Aas
--6G+f
Content-Disposition: form-data; name="email"

gisle@aas.no
--6G+f
Content-Disposition: form-data; name="gender"

M
--6G+f
Content-Disposition: form-data; name="born"

1964
--6G+f
Content-Disposition: form-data; name="init"; filename=".profile"
Content-Type: text/plain

PATH=/local/perl/bin:$PATH
export PATH
--6G+f--
```

Если же Вы установите значение экспортируемой переменной $DYNAMIC_FILE_UPLOAD в некоторое ИСТИННОЕ (TRUE) значение, то получите обратно объект – запрос с подпрограммой в качестве атрибута содержимого. Эта подпрограмма прочтет содержимое любого требуемого файла и возвратит его подходящими порциями. Такая техника позволяет Вам выгружать файлы большого размера на удаленный сервер, не занимая большой объем оперативной памяти. Вы даже можете, если захотите, выгружать бесконечные файлы, типа, */dev/audio*. Другое различие состоит в том, что при использовании такой возможности в запросе не определяется заголовок Content-Length. Такие запросы принимаются далеко не каждым сервером (или серверным приложением).

HTTP::Response
Класс, инкапсулирующий отклики HTTP

Синтаксис

```
require HTTP::Response;
```

Описание

Класс HTTP::Response инкапсулирует HTTP-подобные отклики (ответы). Отклик состоит из строки ответа, нескольких заголовков, и (возможно, пустого) содержимого – контента. Обратите, кстати, внимание на то, что библиотека LWP использует HTTP-подобные отклики и для обработки схем протоколов, отличных от HTTP.

Экземпляры данного класса обычно создаются и возвращаются методом request() объекта LWP::UserAgent:

```
#...
$response = $ua->request($request)
if ($response->is_success) {
    print $response->content;
} else {
    print $response->error_as_HTML;
}
```

Класс HTTP::Response является дочерним классом класса HTTP::Message и, соответственно, наследует все его методы. Наиболее часто используются наследуемые методы header(), push_header(), remove_header(), headers_as_string() и content(). The header convenience methods are also available. Подробная информация – в описании HTTP::Message.

Модулем реализованы следующие методы:

$r = HTTP::Response->new($rc, [$msg, [$header, [$content]]])
: Создает новый объект HTTP::Response , который описывает полученный отклик при помощи кода завершения $rc и необязательного сообщения $msg. Сообщение представляет собой строку текста, которая объясняет полученный код возврата и которая может быть прочитана человеком.

$r->code([$code])
$r->message([$message])
$r->request([$request])
$r->previous([$previousResponse])
: Эти методы обеспечивают внешний доступ к атрибутам объекта. Первые два метода содержат, соответственно, код завершения и текстовое пояснение кода.

  Атрибут request является ссылкой на запрос, который приводит к созданию данного отклика. Это необязательно должен быть тот же запрос, который передается методом $ua->request(), в связи с тем, может в процессе прохождения запроса произойти его перенаправление или проверка авторизации доступа.

  Атрибут previous используется для связи между собой цепочек откликов. Вы получаете цепочку откликов в случае, если первый отклик представляет собой переадресовку (перенаправление запроса) или является несанкционированным.

$r->status_line

Возвращает строку в виде "<код> <сообщение>". Если текст сообщения не определен при помощи параметра, будет подставлено официальное толкование <кода> (смотрите описание модуля HTTP::Status).

$r->base

Возвращает базовый адрес (URL) для данного отклика. Возвращаемое значение – ссылка на объект URI.

Базовый адрес будет получен из одного из следующих источников (в порядке убывания приоритета):

1. Из текста документа HTML, например, из конструкции <BASE HREF="...">.
2. Из заголовка отклика "Content-Base:" или "Content-Location:".

   Для обратной совместимости с более старыми реализациями протокола HTTP метод будет искать в отклике также и заголовок "Base:".

3. Адрес URL приложения, пославшего запрос. Этот адрес может и не быть исходным URL, переданным методом $ua->request(), в связи с тем, что запрос может пройти перед получением адресуемым сервером несколько промежуточных переадресовок (перенаправлений).

Когда модули протокола библиотеки LWP создают объект HTTP::Response, то любой базовый адрес URL, внедренный в текст документа (шаг 1) будет всегда инициализирован значением из заголовка "Content-Base:". Это означает, что данный метод выполняет только 2 последних шага (контент также присутствует не всегда).

$r->as_string

Возвращает текстовое представление отклика. Может быть полезен, главным образом, для целей отладки. Не требует аргументов.

$r->is_info
$r->is_success
$r->is_redirect
$r->is_error

Данные методы возвращают тип отклика: информационный, успешное завершение, перенаправление запроса или сообщение об ошибке.

$r->error_as_HTML()

Возвращает строку, содержащую полный документ HTML, который описывает сообщение об ошибке. Данный метод должен вызываться только в случае, если значение $r->is_error истинно (TRUE).

$r->current_age

Вычисляет "текущий возраст" отклика, как описывается в разделе 13.2.3 спецификации HTTP (rfc2616). Возраст отклика – время с момента отправки отклика сервером, получившим запрос. Возвращает числовое значение – возраст отклика в секундах.

$r->freshness_lifetime

Вычисляет "срок свежести" - время жизни отклика, как описано в разделе 13.2.4. спецификации HTTP (rfc2616). "Срок свежести" – промежуток времени между созданием отклика и его конечным сроком существования. Возвращаемое значение представляет собой число – длительность "срока свежести" в секундах.

Если отклик не содержит заголовка "Expires" или "Cache-Control", то данная функция воспользуется некоторым эвристическим алгоритмом, основанном на значении заголовка 'Last-Modified' для определения соответствующего времени жизни.

$r->is_fresh

Возвращает значение TRUE (ИСТИНА) если отклик является "свежим", основываясь на значениях freshness_lifetime() и current_age(). Если отклик уже не является

"свежим", то он будет повторно запрошен и повторно подтвержден сервером, которому адресован запрос.

$r->fresh_until

Возвращает время окончания "свежести" (срока жизни) объекта.

HTTP::Daemon
класс простого сервера http

Синтаксис

```
use HTTP::Daemon;
use HTTP::Status;
my $d = new HTTP::Daemon;
print "Свяжитесь со мной, пожалуйста, по адресу: <URL:", $d->url, ">\n";
while (my $c = $d->accept) {
    while (my $r = $c->get_request) {
        if ($r->method eq 'GET' and $r->url->path eq "/xyzzy") {
            # помните, такая практика использования  *НЕ* рекомендуется :-)
            $c->send_file_response("/etc/passwd");
        } else {
            $c->send_error(RC_FORBIDDEN)
        }
    }
    $c->close;
    undef($c);
}
```

Описание

Экземпляры класса *HTTP::Daemon* это серверы HTTP/1.1, которые слушают сокет в ожидании поступления входящих запросов. Класс *HTTP::Daemon* – наследованный класс *IO::Socket::INET*, так что Вы также можете выполнять операции с сокетами непосредственно над объектом *IO::Socket::INET*.

Метод accept() method will return when a connection from a client is available. Возвращаемое значение будет ссылкой на объект класса *HTTP::Daemon::ClientConn,* который, в свою очередь, является наследником класса *IO::Socket::INET*. Вызов метода get_request() для данного объекта прочтет данные клиента и возвратит ссылку на объект *HTTP::Request*.

Данный демон HTTP не может реализовать для Вас ветвление процессов. Ваше приложение, т.е. пользователь *HTTP::Daemon* по своему желанию может организовать ветвление. Заметьте также, что пользователь отвечает за генерацию откликов, соответствующих спецификации протокола HTTP/1.1. Класс *HTTP::Daemon::ClientConn* обеспечивает некоторые методы, которые несколько облегчают такую реализацию.

Методы

Ниже приведен список методов, новых или улучшенных по сравнению с методами базового класса *IO::Socket::INET*.

$d = new HTTP::Daemon

Конструктор принимает те же параметры, что и конструктор базового метода *IO::Socket::INET*. Он также может быть вызван вообще без параметров. После запуска демон организует очередь ожидания запросов на 5 соединений и резервирует для

приема некоторый произвольно выбранный номер порта. Сервер, который хочет привязаться к определенному адресу на стандартном порту HTTP может быть запущен, например, так:

```
  $d = new HTTP::Daemon
        LocalAddr => 'www.someplace.com',
        LocalPort => 80;
```

$c = $d->accept([$pkg])

Этот метод подобен аналогичному методу *IO::Socket::accept* но по умолчанию возвращает ссылку на *HTTP::Daemon::ClientConn*. Он возвращает undef, если Вы инициализировали таймаут, а соединение не произошло в течение указанного времени.

$d->url

Возвращает строку URL, которая используется для указания корневого каталога сервера.

$d->product_tokens

Возвращает имя, используемое сервером для собственной идентификации. Оно представляет собой строку, отправляемую в заголовке отклика *Server*. Основной причиной существования данного метода является тот факт, что классы – наследники могут отменить назначение имени, если им потребуется использование другого имени.

Класс *HTTP::Daemon::ClientConn* также является наследником класса *IO::Socket::INET*. Экземпляры данного класса возвращаются методом accept() класса *HTTP::Daemon*. Реализованы следующие дополнительные методы:

$c->get_request([$headers_only])

Читает данные клиента и преобразовывает их в объект *HTTP::Request*, который затем сам и возвращает. Возвращает undef, если чтение запроса завершилось неудачно. Если метод завершается неудачей, то объект *HTTP::Daemon::ClientConn* ($c) должен быть уничтожен и Вы не должны вызывать данный метод снова. Метод $c->reason может предоставить Вам некоторую информацию о том, почему метод $c->get_request вернул значение undef.

Метод $c->get_request поддерживает типы запросов HTTP/1.1, включая кодированную передачу по частям и самоограниченные (ограниченные специальными ограничителями) многосекционные типы содержимого *multipart/**.

Метод $c->get_request при нормальном функционировании не возвращает ничего, пока запрос целиком не будет получен от клиента. Это может быть не тем результатом, который Вы хотите достичь, если запрос предназначен для выгрузки многомегабайтного файла (при помощи механизма HTTP передачи по частям Вы можете даже обеспечить поддержку передачи бесконечных запросов – например, выгрузку потокового звука в реальном времени). Если Вы передаете значение ИСТИНА (TRUE) в качестве аргумента $headers_only, то $c->get_request возвратит ответ немедленно после анализа заголовков запроса, Вы в двльнейшем сами отвечаете за чтение оставшегося содержимого запроса. Если Вы собираетесь вызвать $c->get_request снова в процессе одного соединения, лучше прочтите правильное количество байт.

$c->read_buffer([$new_value])

Байты, прочитанные, но не использованные методом $c->get_request, помещаются в *буфер чтения*. При следующем вызове метода $c->get_request он вначале обработает байты из буфера перед чтением следующей порции данных из самого сетевого соединения. После того, как метод $c->get_request вернет неопределенное значение, буфер чтения освобождается.
Если Вы управляете чтением содержимого запроса самостоятельно, Вам нужно будет очистить этот буфер, прежде чем Вы продолжите чтение, и Вам необходимо поместить в него данные, не обработанные до сих пор. Вам также понадобится такой буфер, если Вы собираетесь реализовать сервис, подобный *101 Switching Protocols (Переключение протоколов).*

Данный метод всегда возвращает предыдущее содержимое буфера и может, по Вашему желанию, заменить содержимое буфера значением, содержащимся в передаваемом аргументе, если Вы укажете его при вызове метода.

$c->reason

Если метод $c->get_request возвращает undef, то Вы можете получить короткую строку, описывающую причину происшедшего путем вызова метода $c->reason.

$c->proto_ge($proto)

Возвращает ИСТИНУ (TRUE) если клиент объявляет, что работает по протоколу с версией, большей или равной переданному аргументу. Аргументом $proto может быть строка в виде "HTTP/1.1" или просто "1.1".

$c->antique_client

Возвращает ИСТИНУ (TRUE) если клиент отвечает по протоколу HTTP/0.9. Такому клиенту не возвращаются код состояния и заголовки. Данный метод аналогичен вызову !$c->proto_ge("HTTP/1.0").

$c->force_last_request

Убедиться, что $c->get_request не будет пытаться читать дальнейшие запросы за рамками данного соединения. Если Вы генерируете отклик, который не самоограничен (имеет несколько секций, разделенных ограничителями), Вы обязаны объявить об этом при помощи вызова данного метода.

Данный атрибут автоматически включается, если клиент объявляет, что он работает по протоколу HTTP/1.0 или более старому и не включает заголовок "Connection: Keep-Alive". Он также автоматически включается, когда клиент, работающий по протоколу HTTP/1.1 или старше, посылает заголовок запроса "Connection: close".

$c->send_status_line([$code, [$mess, [$proto]]])

Отправляет обратно клиенту строку состояния. Если код $code отсутствует, по умолчанию предполагается код 200. Если сообщение $mess отсутствует, то в отклик вставляется сообщение, соответствующее коду $code. Если отсутствует $proto, используется значение переменной $HTTP::Daemon::PROTO.

$c->send_crlf

Посылает клиенту последовательность CRLF.

$c->send_basic_header([$code, [$mess, [$proto]]])

Отправляет обратно клиенту строку состояния, а также заголовки "Date:" и "Server:". Предполагается, что этот заголовок должен быть продолжен и не заканчивается пустой строкой, состоящей из символов CRLF.

$c->send_response([$res])

Отправляет объект *HTTP::Response* клиенту в качестве отклика. Мы упорно пытаемся убедиться в том, что отклик является самоограниченным, так что соединение остается постоянным для дальнейшего обмена запросами / откликами.

Атрибут – содержимое объекта *HTTP::Response* может быть либо обычной строкой, либо ссылкой на подпрограмму. Если это подпрограмма, то что бы ни программа обратного вызова не возвратила, это будет отправлено назад клиенту в качестве содержимого отклика. Подпрограмма будет вызываться до тех пор, пока она не вернет неопределенное или пустое значение. Если клиент понимает протокол HTTP/1.1, то для отклика будет использован метод кодирования с передачей контента по частям.

$c->send_redirect($loc, [$code, [$entity_body]])

Отправляет отклик – перенаправление обратно клиенту. Адрес ($loc) может представлять собой относительный или абсолютный адрес URL. Параметр $code должен быть одним из кодов состояния перенаправления запроса. По умолчанию используется код "301 Moved Permanently" (страница перемещена на новое место навсегда)

$c->send_error([$code, [$error_message]])

Отправляет клиенту отклик – сообщение об ошибке. Если код $code отсутствует, то отсылается сообщение об ошибке "Bad Request" (неверный запрос). $error_message представляет собой строку сообщения об ошибке, которое вставляется в тело страницы HTML.

$c->send_file_response($filename)

Возвращает отклик с указанным файлом $filename в качестве содержимого. Если файл представляет собой директорию, метод пытается сгенерировать страницу HTML, содержащую список файлов этой директории.

$c->send_file($fd);

Отправляет файл клиенту. Файл может быть либо строкой (которая будет интерпретирована в качестве имени файла), либо ссылкой на дескриптор файла *IO::Handle,* либо шаблоном имени файла.

$c->daemon

Возвращает ссылку на соответствующий объект *HTTP::Daemon*.

HTTP::Status - Коды состояния HTTP

Синтаксис

```
use HTTP::Status;
if ($rc != RC_OK) {
    print status_message($rc), "\n";
}
if (is_success($rc)) { ... }
if (is_error($rc)) { ... }
if (is_redirect($rc)) { ... }
```

Описание

HTTP::Status – это набор процедур библиотеки libwww-perl для определения и классификации кодов состояния HTTP. Коды состояния используются для кодирования сообщения - отклика HTTP. Коды состояния соответствуют кодам, определенным документами RFC 2616 и RFC 2518.

Константы

Следующие функции – константы могут быть использованы в качестве мнемонических имен кодов состояния:

```
RC_CONTINUE                        (100)
RC_SWITCHING_PROTOCOLS             (101)
RC_PROCESSING                      (102)
RC_OK                              (200)
RC_CREATED                         (201)
RC_ACCEPTED                        (202)
RC_NON_AUTHORITATIVE_INFORMATION   (203)
RC_NO_CONTENT                      (204)
RC_RESET_CONTENT                   (205)
RC_PARTIAL_CONTENT                 (206)
RC_MULTI_STATUS                    (207)
RC_MULTIPLE_CHOICES                (300)
RC_MOVED_PERMANENTLY               (301)
RC_FOUND                           (302)
RC_SEE_OTHER                       (303)
RC_NOT_MODIFIED                    (304)
RC_USE_PROXY                       (305)
RC_TEMPORARY_REDIRECT              (307)
RC_BAD_REQUEST                     (400)
RC_UNAUTHORIZED                    (401)
RC_PAYMENT_REQUIRED                (402)
RC_FORBIDDEN                       (403)
RC_NOT_FOUND                       (404)
RC_METHOD_NOT_ALLOWED              (405)
RC_NOT_ACCEPTABLE                  (406)
RC_PROXY_AUTHENTICATION_REQUIRED   (407)
RC_REQUEST_TIMEOUT                 (408)
RC_CONFLICT                        (409)
RC_GONE                            (410)
RC_LENGTH_REQUIRED                 (411)
RC_PRECONDITION_FAILED             (412)
RC_REQUEST_ENTITY_TOO_LARGE        (413)
RC_REQUEST_URI_TOO_LARGE           (414)
RC_UNSUPPORTED_MEDIA_TYPE          (415)
RC_REQUEST_RANGE_NOT_SATISFIABLE   (416)
RC_EXPECTATION_FAILED              (417)
RC_UNPROCESSABLE_ENTITY            (422)
RC_LOCKED                          (423)
RC_FAILED_DEPENDENCY               (424)
RC_INTERNAL_SERVER_ERROR           (500)
RC_NOT_IMPLEMENTED                 (501)
RC_BAD_GATEWAY                     (502)
RC_SERVICE_UNAVAILABLE             (503)
RC_GATEWAY_TIMEOUT                 (504)
RC_HTTP_VERSION_NOT_SUPPORTED      (505)
RC_INSUFFICIENT_STORAGE            (507)
```

Функции

Реализованы следующие дополнительные функции. Большинство из них экспортируется по умолчанию.

status_message($code)
: Функция status_message() преобразует коды состояния в нормальные текстовые сообщения, предназначенные для прочтения человеком. Строка соответствует строке, хранимой в константах, перечисленных выше. Если код $code неизвестен, возвращается undef.

is_info($code)
: Возвращает ИСТИНУ (TRUE) если $code – это *Информационный* код состояния. Данный класс кодов завершения указывает на предварительный отклик, который не должен иметь какого – либо содержимого (контента).

is_success($code)
: Возвращает значение ИСТИНА (TRUE) если $code – код состояния *Успешного завершения*.

is_redirect($code)
: Возвращает ИСТИНУ (TRUE) если $code – код состояния *Перенаправления* запроса. Данный класс кодов состояния означает, что от пользовательского агента требуется выполнение дополнительных действий для завершения процесса формирования запроса.

is_error($code)
: Возвращает ИСТИНУ (TRUE), если $code – код состояния *ОШИБКИ*. Функция возвращает ИСТИНУ в обоих случаях, – и при ошибке клиента, и при ошибке сервера.

is_client_error($code)
: Возвращает ИСТИНУ(TRUE) если $code представляет собой код состояния *Ошибка клиента*. Данный класс кодов состояния предназначен для случаев, в которых программа – клиент сообщает об ошибке.

  Эта функция **НЕ** экспортируется по умолчанию.

is_server_error($code)
: Возвращает ИСТИНУ (TRUE) если $code – код состояния *Ошибки сервера*. Данный класс кодов состояния предусмотрен для случаев, когда сервер осведомлен об ошибочной ситуации или не может выполнить запрос.

  Эта функция **НЕ** экспортируется по умолчанию.

Ошибки

Неправильные названия выбраны для неэкспортируемых функций - @EXPORT_OK и @EXPORT – для экспортируемых в начале исходного кода модуля. Сейчас слишком многое экспортируется по умолчанию.

HTTP::Date
процедуры преобраования даты

Синтаксис

```
 use HTTP::Date;
 $string = time2str($time);    # Форматировать Дату/время в виде строки GMT
 $time = str2time($string);    # Преобразовать строку даты в машинное
представление
```

Описание

Этот модуль обеспечивает функции обработки форматов даты, используемых протоколом HTTP (а также некоторыми другими). По умолчанию экспортируются только первые две функции, time2str() и str2time().

time2str([$time])

Функция time2str() преобразует время в машинном формате (количество секунд, прошедших с момента начала эпохи –1970) в строку. Если данная функция вызвана без указания аргумента, в качестве аргумента используется текущее время.

Возвращаемая строка представляется в формате, предпочтительном для протокола HTTP. Такое значение, имеющее фиксированную длину, является подмножеством формата, определенного спецификацией RFC 1123, и представляется в формате универсального времени (GMT). Пример строки времени в таком формате::

```
   Sun, 06 Nov 1994 08:49:37 GMT
```

str2time($str [, $zone])

Функция str2time() преобразует строку во время в машинном формате. Возвращает undef, если формат строки $str не распознан, или значение времени лежит вне допустимого диапазона. Распознаваемые форматы времени аналогичны форматам, обрабатываемым функцией parse_date().

Функция может также получить второй необязательный аргумент, который устанавливает временной пояс, используемый по умолчанию, который используется при преобразовании дат. Данный параметр игнорируется если значение часового пояса присутствует в самой строке времени. Если данный параметр отсутствует, а строка даты не содержит спецификации часового пояса, то в качестве значения числового пояса принимается местный (local) часовой пояс.

Если значение часового пояса не содержит "GMT" или число (подобное "-0800" или "+0100"), то Вам необходимо установить модуль Time::Zone для получения распознаваемой даты.

parse_date($str)

Данная функция пытается анализировать строку даты, а затем возвращает ее в качестве списка числовых значений за которым следует (возможно, неопределенный) указатель часового пояса; ($year, $month, $day, $hour, $min, $sec, $tz). Из параметра $year (год) **не вычитается число 1900**, а числовое значение $month (месяц) **начинается с 1**.

В скалярном контексте числа интерполируются и возвращаются в виде строк в формате "YYYY-MM-DD hh:mm:ss TZ".

Если дата не распознается, возвращается пустой список.

Функция может анализировать следующие форматы:

```
 "Wed, 09 Feb 1994 22:23:32 GMT"       -- формат HTTP
 "Thu Feb  3 17:03:55 GMT 1994"        -- формат ctime(3)
 "Thu Feb  3 00:00:00 1994",           -- формат ANSI C asctime()
 "Tuesday, 08-Feb-94 14:15:29 GMT"     -- устаревший формат HTTP rfc850
 "Tuesday, 08-Feb-1994 14:15:29 GMT"   -- Расширенный формат HTTP rfc850
 "03/Feb/1994:17:03:55 -0700"   -- обычный формат файлов системных log-
                                   журналов
 "09 Feb 1994 22:23:32 GMT"     -- формат HTTP (без дня недели)
 "08-Feb-94 14:15:29 GMT"       -- формат rfc850(без дня недели)
 "08-Feb-1994 14:15:29 GMT"     -- расширенный формат rfc850 (без дня
недели)
 "1994-02-03 14:15:29 -0100"    -- формат ISO 8601
```

```
"1994-02-03 14:15:29"          -- часовой пояс не указан
"1994-02-03"                   -- только дата
"1994-02-03T14:15:29"          -- В качестве разделителя используется T
"19940203T141529Z"             -- компактный формат ISO 8601
"19940203"                     -- только дата
"08-Feb-94"         -- старый формат HTTP rfc850 (без дня недели и времени)
"08-Feb-1994"       -- расширенный формат HTTP rfc850 (без дня недели и
                       времени)
"09 Feb 1994"       -- новый HTTP формат, предлагаемый в качестве стандарта
                       (без дня недели и времени)
"03/Feb/1994"       -- обычный формат файла системного журнала log(без
                       указания времени и часового пояса)
"Feb  3  1994"      -- формат вывода команды  Unix 'ls -l'
"Feb  3 17:03"      -- формат вывода команды  Unix 'ls -l'
"11-15-96  03:52PM" -- формат вывода команды  Windows 'dir'
```

Анализатор игнорирует начальные и концевые пробелы. Он также допускает отсутствие секунд и числовое представление месяцев в большинстве форматов.

Если год опущен, то предполагается, что дата – первая соответствующая дата *в предшествующем* месяце. Если год представлен только 2 цифрами, то parse_date() выберет век, который соответствует году, ближайшему к текущей дате.

time2iso([$time])
Метод, аналогичен time2str(), но возвращает строку в формате "YYYY-MM-DD hh:mm:ss", представляющую время в местном часовом поясе.

time2isoz([$time])
Метод, аналогичен time2str(), но возвращает строку в формате "YYYY-MM-DD hh:mm:ssZ", обозначающую универсальное время.

Смотрите также

time –функция perl, Time::Zone

HTTP::Negotiate
Выбор приемлемого варианта документа

choose – выбор приемлемого варианта документа для обслуживания (при помощи обсуждения контента HTTP)

Синтаксис

```
use HTTP::Negotiate;
#  Идентификатор Качество Тип содержимого Кодирование Набор символов        Язык   Размер
#     ID           qs       Content-Type   Encoding    charset
$variants =
 [['var1',  1.000, 'text/html',   undef,    'iso-8859-1',   'en',    3000],
  ['var2',  0.950, 'text/plain',  'gzip',   'us-ascii',     'no',     400],
  ['var3',  0.3,   'image/gif',   undef,    undef,          undef, 43555],
 ];
@prefered = choose($variants, $request_headers);
$the_one  = choose($variants);
```

Описание

Данный модуль обеспечивает полную реализацию алгоритма обсуждения содержимого (контента) HTTP, описанного в главе 12 спецификации RFC-2616. Обсуждение содержимого позволяет выбрать предпочтительный вариант, основанный на атрибутах согласующих типов данных и значении различных полей заголовков запроса Accept*.

Варианты упорядочиваются по предпочтению при помощи вызова функции choose().

Первый параметр представляет собой ссылку на массив вариантов, между которыми осуществляется выбор. Каждый элемент такого массива представляет собой, в свою очередь, массив, состоящий из элементов [$id, $qs, $content_type, $content_encoding, $charset, $content_language, $content_length] значение которых объясняется ниже. Элементы массива $content_encoding и $content_language могут быть либо одиночными скалярными значениями, либо ссылками на массив, если они состоят из нескольких значений.

Второй необязательный параметр может быть объектом HTTP::Headers, либо объектом HTTP::Request, который ищет заголовки "Accept*". Если этот параметр отсутствует, то спецификация принимаемого содержимого (контента) берется из переменных окружения CGI HTTP_ACCEPT, HTTP_ACCEPT_CHARSET, HTTP_ACCEPT_ENCODING и HTTP_ACCEPT_LANGUAGE.

В контексте массива choose() возвращает список вариантов пар идентификатор / коэффициент качества. Значения сортируются в порядке убывания качества (сначала – документы с большим коэффициентом качества). Если качество двух вариантов одинаково, то документы сортируются по размеру в порядке возрастания (сначала – документы меньшего размера). *Например*:

```
([‘var1’ => 1], [‘var2’, 0.3], [‘var3’ => 0]);
```

Обратите внимание, что в возвращаемый список включаются также документы с нулевым значением коэффициента качества, даже в случае, если они никогда не будут обслужены клиентом.

В скалярном контексте метод возвращает идентификатор варианта с максимальным значением коэффициента качества, или undef , если нет вариантов с ненулевым качеством.

Если переменной $HTTP::Negotiate::DEBUG присвоено значение TRUE (ИСТИНА), то в выходном потоке появится куча мусора, сгенерированного в процессе оценки качества варианта при помощи choose().

Варианты

Вариант описывается списком следующих значений. Если атрибут не имеет смысла или неизвестен варианту, используется значение undef.

identifier
: Строка, используемая в качестве имени варианта. Этот идентификатор возвращается функцией choose() в качестве имени предпочтительного варианта.

qs
: Это число в диапазоне от 0.000 до 1.000, которое описывает "качество источника". Вот что говорит спецификация *RFC-2616* об этом значении:

  Качество источника измеряется поставщиком контента (*сервером - прим. переводчика*) как величина ухудшения качества по сравнению с оригиналом. Например, изображение в формате jpeg будет иметь меньший коэффициент качества ***qs*** при преобразовании в формат XBM, и еще намного меньший ***qs*** при преобразовании в ANSI – графику (*последовательность символов ASCII, которую вы можете увидеть в файлах .nfo и .diz – прим. переводчика*). Обратите, однако, внимание, что качество – свойство источника: оригинал в формате ANSI-графики может потерять в качестве при отображении в формате jpeg. Значения qs должны быть присвоены каждому варианту поставщиком контента; Если значение не установлено, по умолчанию обычно предполагается qs=1.

content-type

Тип содержимого варианта Тип содержимого не включает в себя атрибут charset (кодовая таблица), однако, может содержать другие параметры. Примеры:

```
text/html
text/html;version=2.0
text/plain
image/gif
image/jpg
```

content-encoding

Одна или более кодировок содержимого, которое применялось к варианту. Кодировка обычно используется в качестве модификатора типа содержимого (контента). Наиболее часто встречающимися видами кодировки являются:

```
gzip
compress
```

content-charset

Языковая кодировка, которая используется для случая, когда вариант содержит текст. Значением кодировки, обычно, бывает либо undef , либо одно из следующих:

```
us-ascii
iso-8859-1 ... iso-8859-9
iso-2022-jp
iso-2022-jp-2
iso-2022-kr
unicode-1-1
unicode-1-1-utf-7
unicode-1-1-utf-8
koi8-r
cp866
windows-1251
```

content-language

Описывает один или более языков, которые использовались данным вариантом. Данное свойство описывает один или более языков, которые используются в данном варианте. Спецификация RFC 2616 определяет язык так: В данном контексте язык – это естественный способ коммуникации, при помощи которого люди говорят, пишут или передают каким-либо другим образом информацию другим людям. Компьютерные языки полностью исключаются из этого толкования.

Языковые тэги определены в спецификации RFC-1766. Примеры:

```
no              норвежский
en              международный английский
en-US           американский английский
en-cockney      не вполне английский (кокни ☺)
```

content-length

Количество байт в содержимом (контенте).

Заголовки ACCEPT

Следующие заголовки Accept* могут использоваться для описания, какому содержимому отдается предпочтение для данного запроса. (Это описание является отредактированной выдержкой из спецификации HTTP RFC-2616):

Accept

Данный заголовок может использоваться для того, чтобы указать диапазон допустимых типов содержимого, которое будет получено в ответ на отправленный запрос. Символ "*" используется для группирования типов содержимого в диапазоны; "*/*" означает любые типы контента, а "тип/*" означает все подтипы данного типа.

Параметр q используется для указания фактора качества, который представляет собой числовое выражение предпочтительности для пользователя данного диапазона типов

содержимого (контента). Параметр mbx определяет максимально допустимый размер содержимого отклика. Значения по умолчанию:: q=1 и mbx=infinity (бесконечность). Если заголовок Accept отсутствует, это означает, что клиент принимает содержимое всех типов любого качества (q=1).

Например:

```
Accept: audio/*;q=0.2;mbx=200000, audio/basic
```

означает: "Я предпочитаю документы типа audio/basic (любого размера), но присылайте мне любые типы аудио, если их можно еще слушать после 80% потери качества и размер которых не превышает 200000 байт".

Accept-Charset

Используется для указания, какая именно кодовая языковая таблица допустима для данного запроса. Предполагается, что кодировка "us-ascii" принимается всеми пользовательскими агентами. Если поле Accept-Charset не задано, то по умолчанию предполагается, что допустима любая кодовая таблица. Пример:

```
Accept-Charset: iso-8859-1, unicode-1-1
```

Accept-Encoding

Ограничивает кодировку содержимого (Content-Encoding) значениями, допустимыми для отклика. Если поле Accept-Encoding отсутствует, сервер может предположить, что клиент принимает содержимое с любым кодированием. Пустое значение параметра Accept-Encoding означает, что кодированное содержимое не принимается. Пример:

```
Accept-Encoding: compress, gzip
```

Accept-Language

Данное поле подобно полю Accept, за исключением того, что для запроса ограничивается набор языков, которым представлено содержимое отклика. *(Например, данный документ представлен двумя языками – американским английским и русским – Прим. переводчика)*. Каждому языку может быть поставлен в соответствие коэффициент качества, который означает оценку понимания пользователем данного языка. Например:

```
Accept-Language: no, en-gb;q=0.8, de;q=0.55
```

означает: "Я предпочитаю норвежский, но пойму и британский английский (понимание 80%) и немецкий (понимание 55%)".

HTTP::Cookies
Обработка и хранение файлов cookie

Синтаксис

```
use HTTP::Cookies;
$cookie_jar = HTTP::Cookies->new;
$cookie_jar->add_cookie_header($request);
$cookie_jar->extract_cookies($response);
```

Описание

Файлы cookie – это общий механизм, который может быть использован сервером, как для хранения, так и для извлечения информации клиентской стороны. Подробную информацию о файлах cookie смотрите на <http://www.netscape.com/newsref/std/cookie_spec.html> и <http://www.cookiecentral.com/>. Данный модуль также поддерживает и файлы cookie нового поколения, описанные в документе *draft-ietf-http-state-man-mec-08.txt*. Оба варианта cookie способны (и обязаны) успешно сосуществовать.

Экземпляры класса *HTTP::Cookies* способны хранить наборы Set-Cookie2: и Set-Cookie: заголовков, а также могут использовать эту информацию для инициализации заголовков Cookie в объектах *HTTP::Request*. Состояние объектов *HTTP::Cookies* может быть сохранено в файлах и восстановлено из файлов.

Методы

Реализованы следующие методы:

$cookie_jar = HTTP::Cookies->new;

Конструктор принимает параметры в виде хэша. Распознаются следующие параметры:

```
file:            имя файла, в котором сохраняется и из которого
                 восстанавливается информация cookie
autosave:        сохранить при уничтожении (логический параметр)
ignore_discard:  сохранять даже файлы cookie, которые должны быть уничтожены
                 (логический параметр)
```

В данной версии еще не реализованы, но в дальнейшем могут появиться следующие параметры:

```
  max_cookies               300
  max_cookies_per_domain    20
  max_cookie_size           4096
  no_cookies   список доменов, которым мы никогда не возвращаем cookie
```

$cookie_jar->add_cookie_header($request);

Метод add_cookie_header() устанавливает надлежащий заголовок Cookie: для объекта *HTTP::Request,* переданного в качестве аргумента. Параметру $request должен быть присвоен действительный адрес url до вызова данного метода.

$cookie_jar->extract_cookies($response);

Метод extract_cookies() будет искать заголовки Set-Cookie: Set-Cookie2: в объекте *HTTP::Response,* переданном методу в качестве аргумента. Любой из найденных заголовков используется для обновления состояния $cookie_jar.

$cookie_jar->set_cookie($version, $key, $val, $path, $domain, $port, $path_spec, $secure, $maxage, $discard, \%rest)

Метод set_cookie() обновляет состояние $cookie_jar. Аргументы $key, $val, $domain, $port и $path представляют собой строки. Аргументы $path_spec, $secure, $discard это логические значения. Значение $maxage это количество секунд, в течение которого будет существовать cookie (возраст). Значение <= 0 удаляет данный cookie. %rest определяет прочие различные атрибуты, например, "Comment" и "CommentURL".

$cookie_jar->save([$file]);

Этот метод сохраняет в файле текущее состояние $cookie_jar. Это состояние может быть в дальнейшем восстановлено при помощи метода load(). Если имя файла не указано, будет использовано имя, указанное при создании объекта. Если установлен атрибут *ignore_discard*, то cookie будут сохранены даже в случае, если они были помечены для удаления.

По умолчанию метод сохраняет последовательность строк в формате "Set-Cookie3". "Set-Cookie3'' – это собственный формат LWP, причем неизвестно, совместим ли он со всеми браузерами. Дочерний класс *HTTP::Cookies::Netscape* может быть использован для сохранения файлов cookie в формате, совместимом с Netscape.

$cookie_jar->load([$file]);

Данный метод читает информацию cookie из файла и добавляет ее в $cookie_jar. Файл должен быть в формате, записанном с помощью метода save().

$cookie_jar->revert;

Метод очищает переменную $cookie_jar, а затем заполняет ее содержимым последнего сохраненного файла.

$cookie_jar->clear([$domain, [$path, [$key]]]);

Вызов этого метода без параметров приводит к полной очистке переменной $cookie_jar. Если указывается единственный аргумент- домен, будет удалена только информация cookie, соответствующая указанному домену. При указании второго аргумента удаляются только cookie с определенным путем $path данного домена. При передаче трех аргументов удаляются только cookie с указанным ключом, относящихся к данному домену и указанному пути.

$cookie_jar->scan(\&callback);

Аргументом метода является подпрограмма, которая вызывается для каждого cookie, имеющегося в $cookie_jar. Подпрограмма может быть вызвана со следующими аргументами:

```
 0  версия
 1  ключевое слово
 2  значение
 3  адрес
 4  домен
 5  порт
 6  путь
 7  безопасность
 8  сохраняется до…
 9  отвергнуть
10  хэш
```

$cookie_jar->as_string([$skip_discard]);

Метод as_string() возвращает состояние $cookie_jar в виде последовательности строк заголовков "Set-Cookie3', разделенных символами новой стоки "\n". Если значением $skip_discard является TRUE, он не возвращает строки, соответствующие cookie с атрибутом *Discard*.

Дочерние классы

Мы также разработали дочерний класс с именем *HTTP::Cookies::Netscape* , который загружает и сохраняет файлы cookie в Netscape – совместимом формате. Для того чтобы LWP была способна работать с файлами cookie, совместимыми с Netscape, Вам нужно записать конструктор следующим образом:

```
$cookie_jar = HTTP::Cookies::Netscape->new(
                  File     => "$ENV{HOME}/.netscape/cookies",
                  AutoSave => 1,
              );
```

Обратите, пожалуйста, внимание на то, что формат файлов cookie от Netscape не позволяет сохранить всю информацию из заголовков Set-Cookie2, так что Вы, скорее всего, потеряете часть информации при сохранении файлов cookie в этом формате.

СОДЕРЖАНИЕ