NAME

HTTP::Response – HTTP style response message

VERSION

version 6.22

SYNOPSIS

Response objects are returned by the **request()** method of the `LWP::UserAgent`:

```
# ...
$response = $ua->request($request);
if ($response->is_success) {
    print $response->decoded_content;
}
else {
    print STDERR $response->status_line, "\n";
}
```

DESCRIPTION

The `HTTP::Response` class encapsulates HTTP style responses. A response consists of a response line, some headers, and a content body. Note that the LWP library uses HTTP style responses even for non-HTTP protocol schemes. Instances of this class are usually created and returned by the **request()** method of an `LWP::UserAgent` object.

`HTTP::Response` is a subclass of `HTTP::Message` and therefore inherits its methods. The following additional methods are available:

`$r` = HTTP::Response->new(`$code`)
`$r` = HTTP::Response->new(`$code`, `$msg`)
`$r` = HTTP::Response->new(`$code`, `$msg`, `$header`)
`$r` = HTTP::Response->new(`$code`, `$msg`, `$header`, `$content`)

Constructs a new `HTTP::Response` object describing a response with response code `$code` and optional message `$msg`. The optional `$header` argument should be a reference to an `HTTP::Headers` object or a plain array reference of key/value pairs. The optional `$content` argument should be a string of bytes. The meanings of these arguments are described below.

`$r` = HTTP::Response->parse(`$str`)

This constructs a new response object by parsing the given string.

`$r->code`
`$r->code( $code )`

This is used to get/set the code attribute. The code is a 3 digit number that encode the overall outcome of an HTTP response. The `HTTP::Status` module provide constants that provide mnemonic names for the code attribute.

`$r->message`
`$r->message( $message )`

This is used to get/set the message attribute. The message is a short human readable single line string that explains the response code.

`$r->header( $field )`
`$r->header( $field => $value )`

This is used to get/set header values and it is inherited from `HTTP::Headers` via `HTTP::Message`. See HTTP::Headers for details and other similar methods that can be used to access the headers.

`$r->content`
`$r->content( $bytes )`

This is used to get/set the raw content and it is inherited from the `HTTP::Message` base class. See HTTP::Message for details and other methods that can be used to access the content.

`$r->decoded_content( %options )`
: This will return the content after any `Content-Encoding` and charsets have been decoded. See HTTP::Message for details.

`$r->request`
`$r->request( $request )`
: This is used to get/set the request attribute. The request attribute is a reference to the request that caused this response. It does not have to be the same request passed to the `$ua->`**request()** method, because there might have been redirects and authorization retries in between.

`$r->previous`
`$r->previous( $response )`
: This is used to get/set the previous attribute. The previous attribute is used to link together chains of responses. You get chains of responses if the first response is redirect or unauthorized. The value is `undef` if this is the first response in a chain.

  Note that the method `$r->`redirects is provided as a more convenient way to access the response chain.

`$r->status_line`
: Returns the string "<code> <message>". If the message attribute is not set then the official name of <code> (see HTTP::Status) is substituted.

`$r->base`
: Returns the base URI for this response. The return value will be a reference to a URI object.

  The base URI is obtained from one the following sources (in priority order):

  1. Embedded in the document content, for instance <BASE HREF="..."> in HTML documents.
  2. A "Content-Base:" or a "Content-Location:" header in the response.

     For backwards compatibility with older HTTP implementations we will also look for the "Base:" header.
  3. The URI used to request this response. This might not be the original URI that was passed to `$ua->`**request()** method, because we might have received some redirect responses first.

  If none of these sources provide an absolute URI, undef is returned.

  When the LWP protocol modules produce the HTTP::Response object, then any base URI embedded in the document (step 1) will already have initialized the "Content-Base:" header. (See "parse_head" in LWP::UserAgent). This means that this method only performs the last 2 steps (the content is not always available either).

`$r->filename`
: Returns a filename for this response. Note that doing sanity checks on the returned filename (eg. removing characters that cannot be used on the target filesystem where the filename would be used, and laundering it for security purposes) are the caller's responsibility; the only related thing done by this method is that it makes a simple attempt to return a plain filename with no preceding path segments.

  The filename is obtained from one the following sources (in priority order):

  1. A "Content-Disposition:" header in the response. Proper decoding of RFC 2047 encoded filenames requires the `MIME::QuotedPrint` (for "Q" encoding), `MIME::Base64` (for "B" encoding), and `Encode` modules.
  2. A "Content-Location:" header in the response.
  3. The URI used to request this response. This might not be the original URI that was passed to `$ua->`**request()** method, because we might have received some redirect responses first.

  If a filename cannot be derived from any of these sources, undef is returned.

`$r->as_string`
`$r->as_string( $eol )`
Returns a textual representation of the response.

`$r->is_info`
`$r->is_success`
`$r->is_redirect`
`$r->is_error`
`$r->is_client_error`
`$r->is_server_error`
These methods indicate if the response was informational, successful, a redirection, or an error. See HTTP::Status for the meaning of these.

`$r->error_as_HTML`
Returns a string containing a complete HTML document indicating what error occurred. This method should only be called when `$r->`is_error is TRUE.

`$r->redirects`
Returns the list of redirect responses that lead up to this response by following the `$r->`previous chain. The list order is oldest first.

In scalar context return the number of redirect responses leading up to this one.

`$r->current_age`
Calculates the ''current age'' of the response as specified by RFC 2616 section 13.2.3. The age of a response is the time since it was sent by the origin server. The returned value is a number representing the age in seconds.

`$r->freshness_lifetime( %opt )`
Calculates the ''freshness lifetime'' of the response as specified by RFC 2616 section 13.2.4. The ''freshness lifetime'' is the length of time between the generation of a response and its expiration time. The returned value is the number of seconds until expiry.

If the response does not contain an ''Expires'' or a ''Cache-Control'' header, then this function will apply some simple heuristic based on the ''Last-Modified'' header to determine a suitable lifetime. The following options might be passed to control the heuristics:

heuristic_expiry => `$bool`
If passed as a FALSE value, don't apply heuristics and just return `undef` when ''Expires'' or ''Cache-Control'' is lacking.

h_lastmod_fraction => `$num`
This number represent the fraction of the difference since the ''Last-Modified'' timestamp to make the expiry time. The default is `0.10`, the suggested typical setting of 10% in RFC 2616.

h_min => `$sec`
This is the lower limit of the heuristic expiry age to use. The default is `60` (1 minute).

h_max => `$sec`
This is the upper limit of the heuristic expiry age to use. The default is `86400` (24 hours).

h_default => `$sec`
This is the expiry age to use when nothing else applies. The default is `3600` (1 hour) or ''h_min'' if greater.

`$r->is_fresh( %opt )`
Returns TRUE if the response is fresh, based on the values of **freshness_lifetime()** and **current_age()**. If the response is no longer fresh, then it has to be re-fetched or re-validated by the origin server.

Options might be passed to control expiry heuristics, see the description of **freshness_lifetime()**.

`$r->`fresh_until(`%opt`)
Returns the time (seconds since epoch) when this entity is no longer fresh.

Options might be passed to control expiry heuristics, see the description of **freshness_lifetime**().

SEE ALSO

HTTP::Headers, HTTP::Message, HTTP::Status, HTTP::Request

AUTHOR

Gisle Aas <gisle@activestate.com>

COPYRIGHT AND LICENSE