NAME

Dancer::Request − interface for accessing incoming requests

VERSION

version 1.3140

DESCRIPTION

This class implements a common interface for accessing incoming requests in a Dancer application.

In a route handler, the current request object can be accessed by the `request` method, like in the following example:

```
get '/foo' => sub {
    request->params; # request, params parsed as a hash ref
    request->body; # returns the request body, unparsed
    request->path; # the path requested by the client
    # ...
};
```

A route handler should not read the environment by itself, but should instead use the current request object.

PUBLIC INTERFACE

new()

The constructor of the class, used internally by Dancer's core to create request objects.

It uses the environment hash table given to build the request object:

```
Dancer::Request->new(env => \%ENV);
```

It also accepts the `is_forward` boolean flag, if the new request object is the result of a forward.

init()

Used internally to define some default values and parse parameters.

new_for_request($method, `$path`**,** `$params`**,** `$body`**,** `$headers`**)**

An alternate constructor convenient for test scripts which creates a request object with the arguments given.

forward($request, `$new_location`**)**

Create a new request which is a clone of the current one, apart from the path location, which points instead to the new location. This is used internally to chain requests using the forward keyword.

Note that the new location should be a hash reference. Only one key is required, the `to_url`, that should point to the URL that forward will use. Optional values are the key `params` to a hash of parameters to be added to the current request parameters, and the key `options` that points to a hash of options about the redirect (for instance, `method` pointing to a new request method).

is_forward

Flag that will be set to true if the request has been forwarded.

to_string()

Return a string representing the request object (eg: `"GET /some/path"`)

method()

Return the HTTP method used by the client to access the application.

While this method returns the method string as provided by the environment, it's better to use one of the following boolean accessors if you want to inspect the requested method.

address()

Return the IP address of the client.

remote_host()

Return the remote host of the client. This only works with web servers configured to do a reverse DNS lookup on the client's IP address.

protocol()

Return the protocol (HTTP/1.0 or HTTP/1.1) used for the request.

port()
Return the port of the server.

uri()
An alias to *request_uri()*

request_uri()
Return the raw, undecoded request URI path.

user()
Return remote user if defined.

script_name()
Return script_name from the environment.

scheme()
Return the scheme of the request

secure()
Return true of false, indicating whether the connection is secure

is_get()
Return true if the method requested by the client is 'GET'

is_head()
Return true if the method requested by the client is 'HEAD'

is_patch()
Return true if the method requested by the client is 'PATCH'

is_post()
Return true if the method requested by the client is 'POST'

is_put()
Return true if the method requested by the client is 'PUT'

is_delete()
Return true if the method requested by the client is 'DELETE'

path()
Return the path requested by the client.

base()
Returns an absolute URI for the base of the application. Returns a URI object (which stringifies to the URL, as you'd expect).

uri_base()
Same thing as `base` above, except it removes the last trailing slash in the path if it is the only path.

This means that if your base is *http://myserver/*, `uri_base` will return *http://myserver* (notice no trailing slash). This is considered very useful when using templates to do the following thing:

```
<link rel="stylesheet" href="<% request.uri_base %>/css/style.css" />
```

uri_for(path, params)
Constructs a URI from the base and the passed path. If params (hashref) is supplied, these are added to the query string of the uri. If the base is `http://localhost:5000/foo`, `request->uri_for('/bar', { baz => 'baz' })` would return `http://localhost:5000/foo/bar?baz=baz`. Returns a URI object (which stringifies to the URL, as you'd expect).

params($source)
Called in scalar context, returns a hashref of params, either from the specified source (see below for more info on that) or merging all sources.

So, you can use, for instance:

```
my $foo = params->{foo}
```

If called in list context, returns a list of key => value pairs, so you could use:

```
my %allparams = params;
```

If the incoming form data contains multiple values for the same key, they will be returned as an arrayref.

Fetching only params from a given source

If a required source isn't specified, a mixed hashref (or list of key value pairs, in list context) will be returned; this will contain params from all sources (route, query, body).

In practical terms, this means that if the param `foo` is passed both on the querystring and in a POST body, you can only access one of them.

If you want to see only params from a given source, you can say so by passing the `$source` param to `params()`:

```
my %querystring_params = params('query');
my %route_params       = params('route');
my %post_params        = params('body');
```

If source equals `route`, then only params parsed from the route pattern are returned.

If source equals `query`, then only params parsed from the query string are returned.

If source equals `body`, then only params sent in the request body will be returned.

If another value is given for `$source`, then an exception is triggered.

Vars

Alias to the `params` accessor, for backward-compatibility with `CGI` interface.

request_method

Alias to the `method` accessor, for backward-compatibility with `CGI` interface.

input_handle

Alias to the PSGI input handle (`<request->env->{psgi.input}>`)

content_type()

Return the content type of the request.

content_length()

Return the content length of the request.

header($name)

Return the value of the given header, if present. If the header has multiple values, returns an the list of values if called in list context, the first one in scalar.

headers()

Returns the HTTP::Header object used to store all the headers.

body()

Return the raw body of the request, unparsed.

If you need to access the body of the request, you have to use this accessor and should not try to read `psgi.input` by hand. `Dancer::Request` already did it for you and kept the raw body untouched in there.

is_ajax()

Return true if the value of the header `X-Requested-With` is XMLHttpRequest.

env()

Return the current environment as a hashref.

Note that a request's environment is not always reflected by the global variable `%ENV` (e.g., when running via Plack::Handler::FCGI). In consequence, it is recommended to always rely on the values returned by `env()`, and not to access `%ENV` directly.

uploads()

Returns a reference to a hash containing uploads. Values can be either a Dancer::Request::Upload object, or an arrayref of Dancer::Request::Upload objects.

You should probably use the `upload($name)` accessor instead of manually accessing the `uploads` hash table.

upload($name)

Context-aware accessor for uploads. It's a wrapper around an access to the hash table provided by `uploads()`. It looks at the calling context and returns a corresponding value.

If you have many file uploads under the same name, and call `upload('name')` in an array context, the accessor will unroll the ARRAY ref for you:

```
my @uploads = request->upload('many_uploads'); # OK
```

Whereas with a manual access to the hash table, you'll end up with one element in `@uploads`, being the ARRAY ref:

```
my @uploads = request->uploads->{'many_uploads'}; # $uploads[0]: ARRAY(0xXXXX
```

That is why this accessor should be used instead of a manual access to `uploads`.

Values

Given a request to http://perldancer.org:5000/request-methods?a=1 these are the values returned by the various request-> method calls:

```
base         http://perldancer.org:5000/
host         perldancer.org
uri_base     http://perldancer.org:5000
uri          /request-methods?a=1
request_uri  /request-methods?a=1
path         /request-methods
path_info    /request-methods
method       GET
port         5000
protocol     HTTP/1.1
scheme       http
```

HTTP environment variables

All HTTP environment variables that are in `%ENV` will be provided in the Dancer::Request object through specific accessors, here are those supported:

```
accept
accept_charset
accept_encoding
accept_language
accept_type
agent (alias for user_agent)
connection
forwarded_for_address
forwarded_protocol
forwarded_host
host
keep_alive
path_info
referer
remote_address
request_base
user_agent
```

AUTHORS

This module has been written by Alexis Sukrieh and was mostly inspired by Plack::Request, written by Tatsuiko Miyagawa.

Tatsuiko Miyagawa also gave a hand for the PSGI interface.

LICENCE

This module is released under the same terms as Perl itself.

SEE ALSO

Dancer

AUTHOR

Dancer Core Developers

COPYRIGHT AND LICENSE

This software is copyright (c) 2010 by Alexis Sukrieh.

This is free software; you can redistribute it and/or modify it under the same terms as the Perl 5 programming language system itself.