NAME

CGI::Fast - CGI Interface for Fast CGI

SYNOPSIS

```
use CGI::Fast
socket_path => '9000',
socket_perm => 0777,
listen_queue => 50;

use CGI qw/ :standard /;

$COUNTER = 0;

# optional, will default to STDOUT, STDIN, STDERR
CGI::Fast->file_handles({
fcgi_input_file_handle => IO::Handle->new,
fcgi_output_file_handle => IO::Handle->new,
fcgi_error_file_handle => IO::Handle->new,
});

while ($q = CGI::Fast->new) {
process_request($q);
}
```

DESCRIPTION

CGI::Fast is a subclass of the CGI object created by CGI.pm. It is specialized to work with the FCGI module, which greatly speeds up CGI scripts by turning them into persistently running server processes. Scripts that perform time-consuming initialization processes, such as loading large modules or opening persistent database connections, will see large performance improvements.

OTHER PIECES OF THE PUZZLE

In order to use CGI::Fast you'll need the FCGI module. See http://www.cpan.org/ for details.

WRITING FASTCGI PERL SCRIPTS

FastCGI scripts are persistent: one or more copies of the script are started up when the server initializes, and stay around until the server exits or they die a natural death. After performing whatever one-time initialization it needs, the script enters a loop waiting for incoming connections, processing the request, and waiting some more.

A typical FastCGI script will look like this:

```
#!perl
use CGI::Fast;
do_some_initialization();
while ($q = CGI::Fast->new) {
process_request($q);
}
```

Each time there's a new request, CGI::Fast returns a CGI object to your loop. The rest of the time your script waits in the call to *new()*. When the server requests that your script be terminated, *new()* will return undef. You can of course exit earlier if you choose. A new version of the script will be respawned to take its place (this may be necessary in order to avoid Perl memory leaks in long-running scripts).

CGI.pm's default CGI object mode also works. Just modify the loop this way:

```
while (CGI::Fast->new) {
process_request();
}
```

Calls to *header()*, *start_form()*, etc. will all operate on the current request.

INSTALLING FASTCGI SCRIPTS

See the FastCGI developer's kit documentation for full details. On the Apache server, the following line must be added to srm.conf:

```
AddType application/x-httpd-fcgi .fcgi
```

FastCGI scripts must end in the extension .fcgi. For each script you install, you must add something like the following to srm.conf:

```
FastCgiServer /usr/lib/cgi-bin/file_upload.fcgi -processes 2
```

This instructs Apache to launch two copies of file_upload.fcgi at startup time.

USING FASTCGI SCRIPTS AS CGI SCRIPTS

Any script that works correctly as a FastCGI script will also work correctly when installed as a vanilla CGI script. However it will not see any performance benefit.

EXTERNAL FASTCGI SERVER INVOCATION

FastCGI supports a TCP/IP transport mechanism which allows FastCGI scripts to run external to the webserver, perhaps on a remote machine. To configure the webserver to connect to an external FastCGI server, you would add the following to your srm.conf:

```
FastCgiExternalServer /usr/lib/cgi-bin/file_upload.fcgi -host sputnik:8888
```

Two environment variables affect how the `CGI::Fast` object is created, allowing `CGI::Fast` to be used as an external FastCGI server. (See `FCGI` documentation for `FCGI::OpenSocket` for more information.)

You can set these as ENV variables or imports in the use CGI::Fast statement. If the ENV variables are set then these will be favoured so you can override the import statements on the command line, etc.

FCGI_SOCKET_PATH / socket_path
: The address (TCP/IP) or path (UNIX Domain) of the socket the external FastCGI script to which bind an listen for incoming connections from the web server.

FCGI_SOCKET_PERM / socket_perm
: Permissions for UNIX Domain socket.

FCGI_LISTEN_QUEUE / listen_queue
: Maximum length of the queue of pending connections, defaults to 100.

For example:

```
use CGI::Fast
socket_path => "sputnik:8888",
listen_queue => "50"
;

use CGI qw/ :standard /;

do_some_initialization();

while ($q = CGI::Fast->new) {
process_request($q);
}
```

Or:

```
use CGI::Fast;
use CGI qw/ :standard /;

do_some_initialization();

$ENV{FCGI_SOCKET_PATH} = "sputnik:8888";
```

```
$ENV{FCGI_LISTEN_QUEUE} = 50;

while ($q = CGI::Fast->new) {
process_request($q);
}
```

Note the importance of having use CGI after use CGI::Fast as this will prevent any CGI import pragmas being overwritten by CGI::Fast. You can use CGI::Fast as a drop in replacement like so:

```
use CGI::Fast qw/ :standard /
```

FILE HANDLES

FCGI defaults to using STDIN, STDOUT, and STDERR as its filehandles - this may lead to unexpected redirect of output if you migrate scripts from CGI.pm to CGI::Fast. To get around this you can use the file_handles method, which you must do **before** the first call to CGI::Fast->new. For example using IO::Handle:

```
CGI::Fast->file_handles({
fcgi_input_file_handle => IO::Handle->new,
fcgi_output_file_handle => IO::Handle->new,
fcgi_error_file_handle => IO::Handle->new,
});

while (CGI::Fast->new) {
..
}
```

CAVEATS

I haven't tested this very much.

LICENSE

Address bug reports and comments to:

https://github.com/leejo/cgi-fast

BUGS

This section intentionally left blank.

SEE ALSO

CGI::Carp, CGI