

A Caching CPAN Proxy

by Paul Miller

jettero@cpan.org



The Comprehensive Perl Archive Network (CPAN) is easily the coolest part of Perl. Over the years I've become addicted to so many modules on CPAN that installing a new Linux distribution typically means an hour or two of me installing Perl modules somewhere down the road. Since I maintain a few dozen machines, I find myself installing the same modules over and over again. I also find myself typing the same CPAN shell commands over and over again.

In the CPAN.pm shell, the setting that causes me the most grief is the `urllist`. It's a bit of a project for me to select a mirror site to download the modules from. For me to remember remember which machine had that good mirror, `ssh`-ing over there and copy/pasting it to the new machine quickly becomes tedious.

On my machines, I tend not to update modules when everything is working relatively well. Sometimes many months go by before I open a CPAN shell to install new modules (or update old ones). As a result, I find that each time I open the shell I have to pick a mirror.

I brainstormed one day that I could solve both problems by making my own CPAN mirror. I could simply set all my CPAN shells to point to my mirror and, once they all pointed to the same place, I'd only have to change my mirror settings in one place when I wanted to use a different mirror. I could also pull the same modules over and over from many machines without feeling like I'm wasting public resources.

Initially, I had looked at building an actual mirror. This is usually done with `rsync` and therefore creates a relatively heavy load on the server. It also takes up quite a bit of space (around 1GB), when I really only need a couple megs worth of files. Next, I looked at `CPAN::Mini`. It is definitely closer to what I had in mind, since it only pulls the newest versions and it allows users to filter out namespaces that aren't likely to be needed.

I never did build a mirror with `CPAN::Mini`. I only wanted to mirror the modules I was actually going to use. The problem with filtering is that I don't know know which modules I am going to use because the fifty or so I'm likely to install all have dependencies of their own.

Then it occurred to me that what I really wanted was a simple caching proxy. Furthermore, I wanted this to work via plain old vanilla CGI so I could install it on my shared hosting provider without having to jump through a lot of hoops.

■ My first proxy-----

So, first things first. Using Apache and the `$ENV{PATH_INFO}` environment variable, it's simple enough for me to construct a CGI script that can deal with paths deeper than the CGI:

```
use strict;
use warnings;
use CGI;
use CGI::Carp qw(fatalsToBrowser);

my $cgi = new CGI;

print $cgi->header("text/plain");
print $cgi->path_info, "\n";
```

Now, if I surf to `http://localhost/test.pl/extra1/extra2/extra3`, I get just the portion of the URL after the script name, which is the `path_info`:

```
/extra1/extra2/extra3
```

All I have to do to fetch files from a mirror is re-write the URL and pull the right file, although it would be best if I could pull the MIME type through so the CPAN shell gets the MIME type it expects.

LWP does everything a savvy coder needs for all web-related things, this case being no exception. In just a few lines I can get the HTTP status from the CGI header, the content type (also from the header) and the file contents:

```
use LWP::UserAgent;
use HTTP::Request;

my $mirror = "http://cpan.cs.utah.edu/";
my $ua      = new LWP::UserAgent;

my $path_info = $cgi->path_info;
$path_info =~ s/^\///;

my $new_url = "$mirror$path_info";
my $request = HTTP::Request->new(
    GET => $new_url);
my $response = $ua->request($request);
my $status   = $response->status_line;

print $cgi->header(-status=>$status,
    -type=>$response->header(
        'content-type' ));

if( $response->is_success ) {
    print $response->content;
```