1   ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup

1.1 Synopsis

```
# in startup.pl
use ModPerl::RegistryLoader ();

# explicit uri => filename mapping
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
    debug   => 1, # default 0
);

$rlbb->handler($uri, $filename);

###
# uri => filename mapping using a helper function
sub trans {
    my $uri = shift;
    $uri =~ s|^/registry/|cgi-bin/|;
    return Apache::server_root_relative($uri);
}
my $rl = ModPerl::RegistryLoader->new(
    package => 'ModPerl::Registry',
    trans   => \&trans,
);
$rl->handler($uri);

###
$rlbb->handler($uri, $filename, $virtual_hostname);
```

1.2 Description

This modules allows compilation of scripts, running under packages derived from `ModPerl::RegistryCooker`, at server startup. The script's handler routine is compiled by the parent server, of which children get a copy and thus saves some memory by initially sharing the compiled copy with the parent and saving the overhead of script's compilation on the first request in every httpd instance.

This module is of course useless for those running the `ModPerl::PerlRun` handler, because the scripts get recompiled on each request under this handler.

1.3 Methods

- **new()**

  When creating a new `ModPerl::RegistryLoader` object, one has to specify which of the `ModPerl::RegistryCooker` derived modules to use. For example if a script is going to run under `ModPerl::RegistryBB` the object is initialized as:

  ```
  my $rlbb = ModPerl::RegistryLoader->new(
      package => 'ModPerl::RegistryBB',
  );
  ```

If the package is not specified `ModPerl::Registry` is assumed:

```
my $rlbb = ModPerl::RegistryLoader->new();
```

To turn the debugging on, set the *debug* attribute to a true value:

```
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
    debug   => 1,
);
```

Instead of specifying explicitly a filename for each uri passed to handler(), a special attribute *trans* can be set to a subroutine to perform automatic remapping.

```
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
    trans   => \&trans,
);
```

See the handler() item for an example of using the *trans* attribute.

- **handler()**

  ```
  $rl->handler($uri, [$filename, [$virtual_hostname]]);
  ```

  The handler() method takes argument of `uri` and optionally of `filename` and of `virtual_hostname`.

  URI to filename translation normally doesn't happen until HTTP request time, so we're forced to roll our own translation. If the filename is supplied it's used in translation.

  If the filename is omitted and a `trans` subroutine was not set in new(), the loader will try using the `uri` relative to the `ServerRoot` configuration directive. For example:

  ```
  httpd.conf:
  -----------
  ServerRoot /usr/local/apache
  Alias /registry/ /usr/local/apache/cgi-bin/

  startup.pl:
  -----------
  use ModPerl::RegistryLoader ();
  my $rl = ModPerl::RegistryLoader->new(
      package => 'ModPerl::Registry',
  );
  # preload /usr/local/apache/cgi-bin/test.pl
  $rl->handler(/registry/test.pl);
  ```

  To make the loader smarter about the URI->filename translation, you may provide the `new()` method with a `trans()` function to translate the uri to filename.

The following example will pre-load all files ending with *.pl* in the *cgi-bin* directory relative to `ServerRoot`.

```
httpd.conf:
-----------
ServerRoot /usr/local/apache
Alias /registry/ /usr/local/apache/cgi-bin/

startup.pl:
-----------
{
    # test the scripts pre-loading by using trans sub
    use ModPerl::RegistryLoader ();
    use DirHandle ();
    use strict;

    my $dir = Apache::server_root_relative("cgi-bin");

    sub trans {
        my $uri = shift;
        $uri =~ s|^/registry/|cgi-bin/|;
        return Apache::server_root_relative($uri);
    }

    my $rl = ModPerl::RegistryLoader->new(
        package => "ModPerl::Registry",
        trans   => \&trans,
    );
    my $dh = DirHandle->new($dir) or die $!;

    for my $file ($dh->read) {
        next unless $file =~ /\.pl$/;
        $rl->handler("/registry/$file");
    }
}
```

If `$virtual_hostname` argument is passed it'll be used in the creation of the package name the script will be compiled into for those registry handlers that use *namespace_from_uri()* method. See also the notes on `$ModPerl::RegistryCooker::NameWithVirtualHost` in the `ModPerl::RegistryCooker` documentation.

Also explained in the `ModPerl::RegistryLoader` documentation, this only has an effect at run time if `$ModPerl::RegistryCooker::NameWithVirtualHost` is set to true, otherwise the `$virtual_hostname` argument is ignored.

1.4 Implementation Notes

`ModPerl::RegistryLoader` performs a very simple job, at run time it loads and sub-classes the module passed via the *package* attribute and overrides some of its functions, to emulate the run-time environment. This allows to preload the same script into different registry environments.

1.5 Authors

The original `Apache::RegistryLoader` implemented by Doug MacEachern.

Stas Bekman did the porting to the new registry framework based on `ModPerl::RegistryLoader`.

1.6 SEE ALSO

`ModPerl::RegistryCooker`, `ModPerl::Registry`, `ModPerl::RegistryBB`, `ModPerl::PerlRun`, Apache(3), mod_perl(3)

Table of Contents: