NAME

Catalyst::Manual::Tutorial::02_CatalystBasics - Catalyst Tutorial - Chapter 2: Catalyst Application Development Basics

OVERVIEW

This is **Chapter 2 of 10** for the Catalyst tutorial.

Tutorial Overview

1. Introduction
2. **02_Catalyst Basics**
3. More Catalyst Basics
4. Basic CRUD
5. Authentication
6. Authorization
7. Debugging
8. Testing
9. Advanced CRUD
10. Appendices

DESCRIPTION

In this chapter of the tutorial, we will create a very basic Catalyst web application, demonstrating a number of powerful capabilities, such as:

- Helper Scripts

  Catalyst helper scripts that can be used to rapidly bootstrap the skeletal structure of an application.

- MVC

  Model/View/Controller (MVC) provides an architecture that facilitates a clean "separation of control" between the different portions of your application. Given that many other documents cover this subject in detail, MVC will not be discussed in depth here (for an excellent introduction to MVC and general Catalyst concepts, please see Catalyst::Manual::About). In short:

  - Model

The model usually represents a data store. In most applications, the model equates to the objects that are created from and saved to your SQL database.

    - View

      The view takes model objects and renders them into something for the end user to look at. Normally this involves a template-generation tool that creates HTML for the user's web browser, but it could easily be code that generates other forms such as PDF documents, e-mails, spreadsheets, or even "behind the scenes" formats such as XML and JSON.

    - Controller

      As suggested by its name, the controller takes user requests and routes them to the necessary model and view.

- ORM

  The use of Object-Relational Mapping (ORM) technology for database access. Specifically, ORM provides an automated and standardized means to persist and restore objects to/from a relational database and will automatically create our Catalyst model for use with a database.

You can checkout the source code for this example from the catalyst subversion repository as per the instructions in Catalyst::Manual::Tutorial::01_Intro.

CREATE A CATALYST PROJECT

Catalyst provides a number of helper scripts that can be used to quickly flesh out the basic structure of your application. All Catalyst projects begin with the `catalyst.pl` helper (see Catalyst::Helper for more information on helpers). Also note that as of Catalyst 5.7000, you will not have the helper scripts unless you install both Catalyst::Runtime and Catalyst::Devel.

In this first chapter of the tutorial, use the Catalyst `catalyst.pl` script to initialize the framework for an application called `Hello`:

```
    $ catalyst.pl Hello
    created "Hello"
    created "Hello/script"
    created "Hello/lib"
    created "Hello/root"
    ...
    created "Hello/script/hello_create.pl"
    Change to application directory and Run "perl Makefile.PL" to make sure your install is complete
    $ cd Hello
```

Note: If you are using Strawberry Perl on Win32, drop the ".pl" from the end of the "catalyst.pl" command and simply use "catalyst Hello".

The `catalyst.pl` helper script will display the names of the directories and files it creates:

```
    Changes               # Record of application changes
    lib                   # Lib directory for your app's Perl modules
        Hello             # Application main code directory
            Controller    # Directory for Controller modules
            Model         # Directory for Models
            View          # Directory for Views
        Hello.pm          # Base application module
    Makefile.PL           # Makefile to build application
    hello.conf            # Application configuration file
    README                # README file
    root                  # Equiv of htdocs, dir for templates, css, javascript
        favicon.ico
        static            # Directory for static files
            images        # Directory for image files used in welcome screen
    script                # Directory for Perl scripts
        hello_cgi.pl      # To run your app as a cgi (not recommended)
        hello_create.pl   # To create models, views, controllers
        hello_fastcgi.pl  # To run app as a fastcgi program
        hello_server.pl   # The normal development server
        hello_test.pl     # Test your app from the command line
    t                     # Directory for tests
        01app.t           # Test scaffold
        02pod.t
        03podcoverage.t
```

Catalyst will "auto-discover" modules in the Controller, Model, and View directories. When you use the `hello_create.pl` script it will create Perl module scaffolds in those directories, plus test files in the "t" directory. The default location for templates is in the "root" directory. The scripts in the script directory will always start with the lowercased version of your application name. If your app is MaiTai, then the create script would be "maitai_create.pl".

Though it's too early for any significant celebration, we already have a functioning application. We can use the Catalyst supplied script to start up a development server and view the default Catalyst page in your browser. All scripts in the script directory should be run from the base directory of your application, so change to the Hello directory.

Run the following command to start up the built-in development web server (make sure you didn't forget the "`cd Hello`" from the previous step):

Note: The "-r" argument enables reloading on code changes so you don't have to stop and start the server when you update code. See `perldoc script/hello_server.pl` or `script/hello_server.pl --help` for additional options you might find helpful. Most of the rest of the tutorial will assume that you are using "-r" when you start the development server, but feel free to manually start and stop it (use `Ctrl-C` to breakout of the dev server) if you prefer.

```
    $ script/hello_server.pl -r
    [debug] Debug messages enabled
    [debug] Statistics enabled
    [debug] Loaded plugins:
    .----------------------------------------------------------------------------.
    | Catalyst::Plugin::ConfigLoader  0.30                                       |
    '----------------------------------------------------------------------------'

    [debug] Loaded dispatcher "Catalyst::Dispatcher"
    [debug] Loaded engine "Catalyst::Engine"
    [debug] Found home "/home/catalyst/Hello"
    [debug] Loaded Config "/home/catalyst/Hello/hello.conf"
    [debug] Loaded components:
    .-----------------------------------------------------------------+----------.
    | Class                                                           | Type     |
    +-----------------------------------------------------------------+----------+
    | Hello::Controller::Root                                         | instance |
    '-----------------------------------------------------------------+----------'

    [debug] Loaded Private actions:
    .----------------------+--------------------------------------+--------------.
```

```
| Private              | Class                                 | Method       |
+----------------------+---------------------------------------+--------------+
| /default             | Hello::Controller::Root               | default      |
| /end                 | Hello::Controller::Root               | end          |
| /index               | Hello::Controller::Root               | index        |
'----------------------+---------------------------------------+--------------'

[debug] Loaded Path actions:
.-------------------------------------+--------------------------------------.
| Path                                | Private                              |
+-------------------------------------+--------------------------------------+
| /                                   | /index                               |
| /                                   | /default                             |
'-------------------------------------+--------------------------------------'

[info] Hello powered by Catalyst 5.90002
HTTP::Server::PSGI: Accepting connections at http://0:3000/
```

Point your web browser to http://localhost:3000 (substituting a different hostname or IP address as appropriate) and you should be greeted by the Catalyst welcome screen (if you get some other welcome screen or an "Index" screen, you probably forgot to specify port 3000 in your URL). Information similar to the following should be appended to the logging output of the development server:

```
[info] Hello powered by Catalyst 5.90002
HTTP::Server::PSGI: Accepting connections at http://0:3000/
[info] *** Request 1 (0.067/s) [19026] [Tue Aug 30 17:24:32 2011] ***
[debug] "GET" request for "/" from "192.168.245.2"
[debug] Path is "/"
[debug] Response Code: 200; Content-Type: text/html; charset=utf-8; Content-Length: 5613
[info] Request took 0.040895s (24.453/s)
.------------------------------------------------------------+-----------.
| Action                                                     | Time      |
+------------------------------------------------------------+-----------+
| /index                                                     | 0.000916s |
| /end                                                       | 0.000877s |
'------------------------------------------------------------+-----------'
```

Note: Press `Ctrl-C` to break out of the development server if necessary.

HELLO WORLD

The Simplest Way

The Root.pm controller is a place to put global actions that usually execute on the root URL. Open the `lib/Hello/Controller/Root.pm` file in your editor. You will see the "index" subroutine, which is responsible for displaying the welcome screen that you just saw in your browser.

```
sub index :Path :Args(0) {
    my ( $self, $c ) = @_;

    # Hello World
    $c->response->body( $c->welcome_message );
}
```

Later on you'll want to change that to something more reasonable, such as a "404" message or a redirect, but for now just leave it alone.

The "`$c`" here refers to the Catalyst context, which is used to access the Catalyst application. In addition to many other things, the Catalyst context provides access to "response" and "request" objects. (See Catalyst::Runtime, Catalyst::Response, and Catalyst::Request)

`$c->response->body` sets the HTTP response (see Catalyst::Response), while `$c->welcome_message` is a special method that returns the welcome message that you saw in your browser.

The ":Path :Args(0)" after the method name are attributes which determine which URLs will be dispatched to this method. (You might see ":Private" if you are using an older version of Catalyst, but using that with "default" or "index" is currently deprecated. If so, you should also probably upgrade before continuing the tutorial.)

Some MVC frameworks handle dispatching in a central place. Catalyst, by policy, prefers to handle URL dispatching with attributes on controller methods. There is a lot of flexibility in specifying which URLs to match. This particular method will match all URLs, because it doesn't specify the path (nothing comes after "Path"), but will only accept a URL without any args because of the ":Args(0)".

The default is to map URLs to controller names, and because of the way that Perl handles namespaces through package names, it is simple to create hierarchical structures in Catalyst. This means that you can create controllers with deeply nested actions in a clean and logical way. For example, the URL `http://hello.com/admin/articles/create` maps to the package `Hello::Controller::Admin::Articles`, and the `create` method.

While you leave the `script/hello_server.pl -r` command running the development server in one window (don't forget the "-r" at the end!), open another window and add the following subroutine to your `lib/Hello/Controller/Root.pm` file:

```
    sub hello :Global {
        my ( $self, $c ) = @_;

        $c->response->body("Hello, World!");
    }
```

TIP: See Appendix 1 for tips on removing the leading spaces when cutting and pasting example code from POD-based documents.

Notice in the window running the Development Server that you should get output similar to the following:

```
    Saw changes to the following files:
     - /home/catalyst/Hello/lib/Hello/Controller/Root.pm (modify)

    Attempting to restart the server
    ...
    [debug] Loaded Private actions:
    .----------------------+--------------------------------------+--------------.
    | Private              | Class                                | Method       |
    +----------------------+--------------------------------------+--------------+
    | /default             | Hello::Controller::Root              | default      |
    | /end                 | Hello::Controller::Root              | end          |
    | /index               | Hello::Controller::Root              | index        |
    | /hello               | Hello::Controller::Root              | hello        |
    '----------------------+--------------------------------------+--------------'
    ...
```

The development server noticed the change in `Hello::Controller::Root` and automatically restarted itself.

Go to http://localhost:3000/hello to see "Hello, World!". Also notice that the newly defined 'hello' action is listed under "Loaded Private actions" in the development server debug output.

Hello, World! Using a View and a Template

In the Catalyst world a "View" itself is not a page of XHTML or a template designed to present a page to a browser. Rather, it is the module that determines the *type* of view -- HTML, PDF, XML, etc. For the thing that generates the *content* of that view (such as a Template Toolkit template file), the actual templates go under the "root" directory.

To create a TT view, run:

```
    $ script/hello_create.pl view HTML TT
```

This creates the `lib/Hello/View/HTML.pm` module, which is a subclass of `Catalyst::View::TT`.

- The "view" keyword tells the create script that you are creating a view.
- The first argument "HTML" tells the script to name the View module "HTML.pm", which is a commonly used name for TT views. You can name it anything you want, such as "MyView.pm". If you have more than one view, be sure to set the default_view in Hello.pm (See Catalyst::View::TT for more details on setting this).
- The final "TT" tells Catalyst the *type* of the view, with "TT" indicating that you want to use a Template Toolkit view.

If you look at `lib/Hello/View/HTML.pm` you will find that it only contains a config statement to set the TT extension to ".tt".

Now that the HTML.pm "View" exists, Catalyst will autodiscover it and be able to use it to display the view templates using the "process" method that it inherits from the `Catalyst::View::TT` class.

Template Toolkit is a very full-featured template facility, with excellent documentation at http://template-toolkit.org/, but since this is not a TT tutorial, we'll stick to only basic TT usage here (and explore some of the more common TT features in later chapters of the tutorial).

Create a `root/hello.tt` template file (put it in the `root` under the `Hello` directory that is the base of your application). Here is a simple sample:

```
    <p>
        This is a TT view template, called '[% template.name %]'.
    </p>
```

[% and %] are markers for the TT parts of the template. Inside you can access Perl variables and classes, and use TT directives. In this case, we're using a special TT variable that defines the name of the template file (`hello.tt`). The rest of the template is normal HTML.

Change the hello method in `lib/Hello/Controller/Root.pm` to the following:

```
    sub hello :Global {
        my ( $self, $c ) = @_;
        $c->stash(template => 'hello.tt');
    }
```

This time, instead of doing `$c->response->body()`, you are setting the value of the "template" hash key in the Catalyst "stash", an area for putting information to share with other parts of your application. The "template" key determines which template will be displayed at the end of the request cycle. Catalyst controllers have a default "end" action for all methods which causes the first (or default) view to be rendered (unless there's a `$c->response->body()` statement). So your template will be magically displayed at the end of your method.

After saving the file, the development server should automatically restart (again, the tutorial is written to assume that you are using the "-r" option -- manually restart it if you aren't), and look at http://localhost:3000/hello in your web browser again. You should see the template that you just created.

TIP: If you keep the server running with "-r" in a "background window," don't let that window get totally hidden... if you have a syntax error in your code, the debug server output will contain the error information.

Note: You will probably run into a variation of the "stash" statement above that looks like:

```
    $c->stash->{template} = 'hello.tt';
```

Although this style is still relatively common, the approach we used previous is becoming more common because it allows you to set multiple stash variables in one line. For example:

```
    $c->stash(template => 'hello.tt', foo => 'bar',
              another_thing => 1);
```

You can also set multiple stash values with a hashref:

```
    $c->stash({template => 'hello.tt', foo => 'bar',
              another_thing => 1});
```

Any of these formats work, but the `$c->stash(name => value);` style is growing in popularity -- you may wish to use it all the time (even when you are only setting a single value).

CREATE A SIMPLE CONTROLLER AND AN ACTION

Create a controller named "Site" by executing the create script:

```
    $ script/hello_create.pl controller Site
```

This will create a `lib/Hello/Controller/Site.pm` file (and a test file). If you bring Site.pm up in your editor, you can see that there's not much there to see.

In `lib/Hello/Controller/Site.pm`, add the following method:

```
    sub test :Local {
        my ( $self, $c ) = @_;

        $c->stash(username => 'John',
                  template => 'site/test.tt');
    }
```

Notice the "Local" attribute on the `test` method. This will cause the `test` action (now that we have assigned an "action type" to the method it appears as a "controller action" to Catalyst) to be executed on the "controller/method" URL, or, in this case, "site/test". We will see additional information on controller actions throughout the rest of the tutorial, but if you are curious take a look at "Actions" in Catalyst::Manual::Intro.

It's not actually necessary to set the template value as we do here. By default TT will attempt to render a template that follows the naming pattern "controller/method.tt", and we're following that pattern here. However, in other situations you will need to specify the template (such as if you've "forwarded" to the method, or if it doesn't follow the default naming convention).

We've also put the variable "username" into the stash, for use in the template.

Make a subdirectory "site" in the "root" directory.

```
    $ mkdir root/site
```

Create a new template file in that directory named `root/site/test.tt` and include a line like:

```
    <p>Hello, [% username %]!</p>
```

Once the server automatically restarts, notice in the server output that `/site/test` is listed in the Loaded Path actions. Go to http://localhost:3000/site/test in your browser and you should see your test.tt file displayed, including the name "John" that you set in the controller.

You can jump to the next chapter of the tutorial here: More Catalyst Basics

AUTHORS

Gerda Shank, `gerda.shank@gmail.com` Kennedy Clark, `hkclark@gmail.com`

Feel free to contact the author for any errors or suggestions, but the best way to report issues is via the CPAN RT Bug system at https://rt.cpan.org/Public/Dist/Display.html?Name=Catalyst-Manual.

syntax highlighting: no syntax highlighting ▼