Catalyst & Moo

Hopefully Help Tips

Kennedy Clark

Austin.pm - 140722

Agenda

Kennedy Clark
Vienna, Virginia
kclark@kennedyclark.com

- Catalyst
  - Controllers
  - Models
  - Views
  - Miscellaneous
- Moo *(time permitting)*
  - What is it
  - Common usage patterns
  - General tips

Catalyst

Controllers

Catalyst Controllers

- Quick review - MVC:
  - Controllers receive requests
  - Gets data from your Model
  - Lets a View render it to send back to the browser
- Example Controller Action:

```
package MyApp::Controller::Users;                  # Some Controller class
sub view :Path('/users/view') {                    # Config dispatch logic
    my ($self, $c) = @_;                           # Get $self & cat context
    $c->stash(                                     # Stash is per-request
        users    => [ $c->model('DB::User')->all ], # Get data from Model
        template => 'users/view.tt',               # Specify View template
    );
}
```

Controllers - Use Chaining:

```
package MyApp::Controller::Users;
sub base :Chained('/') :PathPart('users') :CaptureArgs(0) {
   # Check auth here
}
sub viewall :Chained('base') :PathPart('view') :Args(0) {
   # Show list of all users
}
sub get_user :Chained('base') :PathPart('') :CaptureArgs(1) {
   my ($self, $c, $user_id) = @_;
   $c->stash( user_obj => $c->model('DB::User')->find($user_id) );
}
sub view_user :Chained('get_user') :PathPart('view') :Args(0) {
   # Display details on the single user here using user_obj already in stash
}
sub edit_user :Chained('get_user') :PathPart('edit') :Args(0) {
   # Edit details on the single user here using user_obj already in stash
}
```

Not an endpoint

An endpoint

Controllers - Use Chaining:

- View from dev server debug output:

```
$ perl script/myapp_server -rd
...
[debug] Loaded Chained actions:
.--------------------------------------+-------------------------------------.
| Path Spec                            | Private                             |
+--------------------------------------+-------------------------------------+
| /users/view                          | /users/base (0)                     |
|                                      | => /users/viewall                   |
| /users/*/view                        | /users/base (0)                     |
|                                      | -> /users/get_user (1)              |
|                                      | => /users/view_user                 |
| /users/*/edit                        | /users/base (0)                     |
|                                      | -> /users/get_user (1)              |
|                                      | => /users/edit_user                 |
'--------------------------------------+-------------------------------------'
```

Controllers - Configuration:

- myapp.yml:

```
$ cat myapp.yml
---
name: MyApp
Controller::Users:
    corp_hr_api_uri: 'https://somewhere.com/employees'
    corp_hr_api_token: 'Rj#sj4s#fLelrig3jl9'
```

- Note: Can access like this, but don't:

```
$c->config->{Controller::Users}->{corp_hr_api_url}
$c->config->{Controller::Users}->{corp_hr_api_token}

MyApp->config->{Controller::Users}->{corp_hr_api_url}
MyApp->config->{Controller::Users}->{corp_hr_api_token}
```

Controllers - Configuration:

- MyApp::Controller::Users

```
package MyApp::Controller::Users;
use Moose;
use namespace::autoclean;
BEGIN {extends 'Catalyst::Controller'; }   # Boilerplate down to here
has corp_hr_api_url => (                   # Add attributes via normal Moose
   is         => 'ro',
   required   => 1,
   isa        => 'Str',
);
has corp_hr_api_token => (
   is         => 'ro',
   required   => 1,
   isa        => 'Str',
);
```

Controllers - Configuration:

- To use a variable in controller action

```
$c->log->debug( "Using HR API URL: " . $self->corp_hr_api_url );
```

- If you forget to define a variable:

```
$ perl script/myapp_server.pl -r
Couldn't instantiate component "MyApp::Controller::Users", "Attribute
  (corp_hr_api_token) is required at (eval 1095)[(eval 1094)
  [/home/kclark/perl5/lib/perl5/Eval/Closure.pm:123]:3] line 39.
MyApp::Controller::Users::new('MyApp::Controller::Users', 'MyApp', 'HASH
  (0xa8c6a78)') called at /home/kclark/perl5/lib/perl5/Catalyst/Component.pm line
  110
...
```

Models

Catalyst Models

- Represent "data" and business logic for your application
  - Many types:
    - Database
    - Files
    - Web Service (e.g., RESTful API client)
- Goal:
  - "Thin" controllers
  - "Fat" (or at least "Smart") models

Use Catalyst::Model::Adaptor

- Build your model to function *outside* Catalyst
  - Including tests!
  - Just a set of classes
  - Consider Moo (stay tuned)

- Then use Catalyst::Model::Adaptor to create a single "glue class" in MyApp::Model

Catalyst::Model::Adaptor Example

The external model class (or classes):

```
package OutsideMyApp::SomeClass;
use Moo;   # Or any other means of creating a class
sub do_something {
    my ($self) = @_;
    return "test message";
}
1;
```

The adaptor class:

```
package MyApp::Model::SomeClass;
use base 'Catalyst::Model::Adaptor';
__PACKAGE__->config( class => 'OutsideMyApp::SomeClass' );
```

Catalyst::Model::Adaptor Example

Using model from Catalyst:

```
package MyApp::Controller::Root;
...
sub test_page :Path('/test_msg') {
    my ($self, $c) = @_;
    ...
    # Use $c->model to access your model
    $c->response->body( $c->model('SomeClass')->do_something );
}
...
```

Catalyst::Model::Adaptor:

- Default = instantiate at webapp startup
  - What you want most of the time
- Options for:
  - Per Request ("Factory::PerRequest")
  - Per call to $c->model ("Factory")

Models - Use DBIx::Class

- Aka "DBIC"
- Extremely powerful ORM
- E.g. - Chaining for queries - very expressive:

```
my @active_users
    = $c->model('DB::Company')
        ->find($company_id)          # Get the company
        ->users                      # Follow relationship to users
        ->active                     # But only active users
        ->created_in_last_days(30)   # That have been created in last 30 days
        ->with_role('admin');        # That are administrators
```

Models - DBIx::Class - Key Classes:

1. Schema Class
   - Represents the whole database
   - MyApp::Schema.pm
     - From Catalyst: $c->model('DB');
2. Result Classes
   - One per Table (or View)
   - Represents a row of results from a query
   - Add per-row methods/logic here
   - E.g.: MyApp::Schema::Result::*

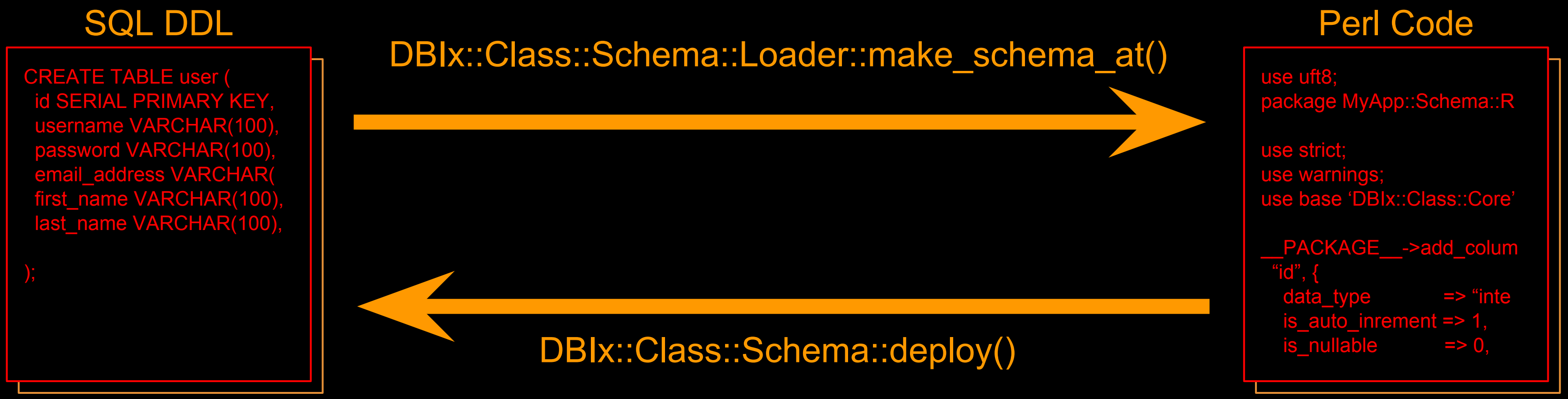
Models - DBIx::Class - Key Classes:

3. ResultSet Classes
    - A set of rows (possibly an entire table) from a query
    - Equates to conditions and joins needed in SQL
    - Add "canned queries" here
    - Eg: MyApp::Schema::ResultSet::*
    - From Catalyst: $c->model("DB::$table_name");

- Be sure to use all 3 to customize your model

Models - DBIx::Class

- 2 options Result Classes ⇔ SQL Schema:


SQL DDL
DBIx::Class::Schema::Loader::make_schema_at()
Perl Code
DBIx::Class::Schema::deploy()

```
CREATE TABLE user (
  id SERIAL PRIMARY KEY,
  username VARCHAR(100),
  password VARCHAR(100),
  email_address VARCHAR(
  first_name VARCHAR(100),
  last_name VARCHAR(100),

);
```

```
use uft8;
package MyApp::Schema::R

use strict;
use warnings;
use base ‘DBIx::Class::Core’

__PACKAGE__->add_colum
  “id”, {
    data_type          => “inte
    is_auto_inrement => 1,
    is_nullable         => 0,
```

Models and Catalyst Context ($c)

- Avoid using the $c in your Models
  - Binds your models to Catalyst - harder to reuse
- But if you do need it:

```
sub ACCEPT_CONTEXT {                  # Catalyst calls this, not you
    my ($self, $c, @args) = @_;
    # Do something with $c
}
```

- Probably want one instance per request, e.g.:
  - Catalyst::Model::Factory::PerRequest
  - Catalyst::Component::InstancePerContext

Views

Catalyst Views

- Normally Template Toolkit
  - Very powerful, easy to use, lots of features, etc.
  - But many other options if you prefer

- Can have multiple views in one app:
  - MyApp::View::TT (default site)
  - MyApp::View::TTNewLook (beta test new layout)
  - MyApp::View::Mobile (lightweight version)

Views - Template Toolkit & DBIC

- In most cases, it’s all the same
- But one thing to watch out for:
  - As a part of DBIx::Class being "smart", it returns:
    - Scalar Context: A ResultSet
    - List Context: The list of objects from the ResultSet
  - TT calls methods in list context

    ```
    [% users.purchases.count %]
    ```

  - If you need a ResultSet, use the _rs variant

    ```
    [% users.purchases_rs.count %]

    [% users.search_rs(...).count %]  (but avoid search() in Views,
              use a ResultSet method for a “canned search” instead)
    ```

Miscellaneous

Misc:

- Use local::lib
  - Using system Perl isn't worth the trouble
  - Can just tar up your development libs and copy to production to have exactly the same versions

- Use Log::Contextual
  - Docs aren't great, but the functionality is
  - Esp. useful in model classes (where you don't have $c->log)

Misc - Debugging

- Normal debugger
  - Set a breakpoint:

```
sub view :Path('view') {
    my ($self, $c) = @_;
$DB::single=1;
    $c->stash( users => [ $c->model('DB::User')->all ] );
}
```

  - Run development server under debugger:
    - perl -d script/myapp_server
    - Note: Do NOT use with -r

Misc - Debugging

- Dumping data to log:

```
sub view :Path('view') {
    my ($self, $c) = @_;
    my @users = $c->model('DB::User')->all;
    $c->stash( users => \@users );
# Temporarily add logging
use Data::Dumper::Concise;
$Data::Dumper::Maxdepth = 3;# DBIC objects are BIG so limit depth
$c->log->debug( 'Users: ' . Dumper( @users ) );
}
```

- DBIx::Class:
  - DBIC_TRACE=1 perl script/myapp_server -r

Misc

- High Availability FastCGI Setup:
    - E.g.: nginix:
        - Create a "upstream fastcgi_pool"
        - Then run two copies of FastCGI inside the pool
            - Each can have as many -n processes as you want
            - Can restart each and it’s normally “hitless”
                - At least for small patches or just a restart

Moo

Moo - What is it?

- "Lightweight Moose"
  - Moose is great, but startup is slow
- Moo:
  - Fast startup
  - Pure Perl
  - All you need for almost all situations:
    - POD says Moo "provides almost -- but not quite -- two thirds of Moose"
    - But it's 95%+ of what you need on a daily basis in my experience

Use Moo

- Catalyst is fairly tightly bound to full Moose
- But nothing says you can't go with a more lightweight option for you model classes
    - You are building them outside Catalyst, right?
- Using Moo, your tests for these external classes will be MUCH faster

Moo - Compatibility

- Moo has many Moose compatibility features
- And TIMTOWTDI definitely applies
- But Moo is fairly opinionated
  - You probably want to embrace a certain style of Moo/Moose syntax and features
    - For Example:
      - Uses MooseX::AttributeShortcuts syntax
      - Types are coderefs vs. strings (or blessed constraint)
      - Can use MooX::late for "plain old Moose" syntax
    - Automatically enabled FATAL warnings

Moo - Common Patterns

```
  package MyClass;
  use Moo;
  use warnings NONFATAL => 'uninitialized';   # Prevent death on undef
  use Types::Standard qw[Str InstanceOf];     # See POD for other types available
  use URI;
1 has scheme => (                          # Example of attribute required by constructor
      is       => 'ro',
      isa      => Str,                     # Note coderef from Types::Standard, not string
      required => 1,
  );
2 has uri => (                             # Example of attrib you build at runtime
      is       => 'lazy',                  # "lazy => 1" like Moose on separate line OK too
      isa      => InstanceOf['URI'],       # Must be an Object of type URI
      builder  => sub {                    # Anon sub to construct attrib value
          URI->new( shift->scheme . '://perl.com');     # Die here if invalid value
      },
  );
```

Most often, you just need one of these 2 options

Moo or Moose - General Tips

- Use lazy
  - Required when accessing other attributes
  - But can give performance boost in other cases
- Use Roles
  - As an alternative to traditional OO inheritance
- Use method modifiers:
  - before, after, around
- Use ‘handles’ for delegation

Thank you