Cathartic Catalytic Conversion

MVC is good for your sanity.

Intro to Catalyst + Tips for App Conversion

Dave Gray
<dave@omniti.com>

We're hiring!
<jobs@omniti.com>
<http://www.omniti.com/people/jobs/>

MVC Quick Summary

Model:

You put your data in here, and expect to get it back out again.

Controller:

Go get or present back data that end users ask for.

View:

Make forms and data shiny and present them intuitively.

MCV? Yeah. They're not in the same order as the acronym here because it's clearer (to me) to think about the different parts as layers, one sitting on top of the next in MCV order.

What is Catalyst?

Catalyst is...

a Perl MVC framework,

glue between other modules

What is Catalyst?

Catalyst is...

a Perl MVC framework,
glue between other modules

a time-saver
like perl, simple things are easy

Getting Catalyst

Matt Trout's cat-install:

<http://www.shadowcatsystems.co.uk/static/cat-install>

DIY:

```
perl -MCPAN -e 'install Task::Catalyst'
perl -MCPAN -e 'install Catalyst::Devel'
```

good step-by-step tutorials

<http://search.cpan.org/perldoc?Catalyst::Manual::Tutorial>

Other Modules

Model $\longrightarrow$ DBIx::Class manages data

Other Modules

$$\begin{tikzcd}
\textit{Controller} \arrow[r] & \text{HTML::Widget} & \text{requests data, organizes data} \\
\textit{Model} \arrow[r] & \text{DBIx::Class} \arrow[u] & \text{manages data}
\end{tikzcd}$$

Other Modules

$$\begin{tikzcd}
\textit{View} \arrow[r] & \text{Template::Toolkit} & \text{presents data} \\
\textit{Controller} \arrow[r] & \text{HTML::Widget} \arrow[u] & \text{requests data, organizes data} \\
\textit{Model} \arrow[r] & \text{DBIx::Class} \arrow[u] & \text{manages data}
\end{tikzcd}$$

Template::Toolkit

<http://search.cpan.org/perldoc?Template::Toolkit>

One of the packages for creating Views with Catalyst

Template::Toolkit

<http://search.cpan.org/perldoc?Template::Toolkit>

One of the packages for creating Views with Catalyst

Most other templating systems are supported:
Mason, HTML::Template, etc

Template::Toolkit

<http://search.cpan.org/perldoc?Template::Toolkit>

One of the packages for creating Views with Catalyst

Most other templating systems are supported:
Mason, HTML::Template, etc

Keep it simple in the View layer, just because you *can* do a lot more there doesn't mean you *have to*.

HTML::Widget

<http://search.cpan.org/perldoc?HTML::Widget>

A good module for generating forms, validating input, and generally avoiding HTML.

HTML::Widget

<http://search.cpan.org/perldoc?HTML::Widget>

A good module for generating forms, validating input, and generally avoiding HTML.

Easy to use from your templating library, can customize the output HTML if you don't like the defaults.

HTML::Widget

<http://search.cpan.org/perldoc?HTML::Widget>

A good module for generating forms, validating input, and generally avoiding HTML.

Easy to use from your templating library, can customize the output HTML if you don't like the defaults.

The default validation constraints cover 95% of the input types you will encounter.

DBIx::Class

<http://search.cpan.org/perldoc?DBIx::Class>

Provides an Object Relational Mapping (ORM) layer

DBIx::Class

<http://search.cpan.org/perldoc?DBIx::Class>

Provides an Object Relational Mapping (ORM) layer

Abstracts away storage-specific details

DBIx::Class

<http://search.cpan.org/perldoc?DBIx::Class>

Provides an Object Relational Mapping (ORM) layer

Abstracts away storage-specific details

Performs Create, Read, Update, Delete (CRUD) operations from perl data structures.

DBIx::Class

<http://search.cpan.org/perldoc?DBIx::Class>

Provides an Object Relational Mapping (ORM) layer

Abstracts away storage-specific details

Performs Create, Read, Update, Delete (CRUD) operations from perl data structures.

DBIx::Class::HTMLWidget uses form objects to load data from and store data in your database.

Where's Catalyst?

Template::Toolkit

DBIx::Class

HTML::Widget

Where’s Catalyst?

HTML::Widget::Filter Template::Toolkit
DBIx::Class::ResultSet
HTML::Widget::Result
DBIx::Class
DBIx::Class::ResultSource
HTML::Widget::Element HTML::Widget::Constraint
HTML::Widget
DBIx::Class::HTMLWidget
DBIx::Class::Schema

Where's Catalyst?

omniti

Where's Catalyst?

(obligatory lolcat.)

Catalyst::Plugin::StackTrace Catalyst::Helper::View::TTSite

Catalyst::Plugin::Session Catalyst

Catalyst::Model::DBIC::Schema

Catalyst::Helper::View::TT Catalyst::Devel

Where's Catalyst?

(obligatory lolcat.)

Here's Catalyst!

omniti

Helpers:

Speed up development by generating functional skeletons that can be extended as desired

Here's Catalyst!

omniti

Helpers:

Speed up development by generating functional skeletons that can be extended as desired

Plugins:

Easy integration with modules that allows you to mix and match to get the environment you work best in

Here's Catalyst!

Helpers:

Speed up development by generating functional skeletons that can be extended as desired

Plugins:

Easy integration with modules that allows you to mix and match to get the environment you work best in

Here's Catalyst!

Servers:

Standalone for easy testing, CGI and FastCGI for deployment with better performance (works with mod_perl too)

Catalyst is the glue

It really ties the room together

Application Layout

Converting existing apps to Catalyst

1. Make your data accessible for CRUD

   A) Basic Catalyst setup. Run helpers to generate directory structures and sample modules.

   B) Write the classes that enable DBIx::Class to access and manage your data source. Here we are basically summarizing the DDL in a way that DBIx::Class can understand.

   C) Tell DBIx::Class how to get to your data.

Application Layout

Converting existing apps to Catalyst

Here comes the code!

Application Layout

Converting existing apps to Catalyst

A) Create A Catalyst Application

(Real Estate Information Service)

```
$ catalyst.pl REIS
... creates a bunch of files
$ cd REIS/
```

Application Layout
Converting existing apps to Catalyst

A) Create A Catalyst Application

(Real Estate Information Service)

```
$ catalyst.pl REIS
... creates a bunch of files
$ cd REIS/
```

Make a Holding Pen for your Model classes

```
$ mkdir lib/REISDB
```

Application Layout

Converting existing apps to Catalyst

B) Write the Classes

a table in your database becomes a module in your model

```
CREATE TABLE InterestRates1 (
    InterestRateID int auto_increment,
    Year int(4) not null,
    Month tinyint(3) unsigned not null,
    Rate decimal(6,4) unsigned not null,
    Points decimal(2,1) unsigned not null,
    ExcelDate int(10),
    PRIMARY KEY ( InterestRateID ),
    UNIQUE (Year, Month)
);
```

```
# REIS/lib/REISDB/InterestRate.pm

package REISDB::InterestRate;
use base 'DBIx::Class';
__PACKAGE__->load_components( qw(
    PK::Auto
    Core
    HTMLWidget
));
__PACKAGE__->table( 'InterestRates' );
__PACKAGE__->add_columns( qw(
 InterestRateID
 Year
 Month
 Rate
 Points
 ExcelDate
));
__PACKAGE__->set_primary_key( qw(InterestRateID) );

1;
```

Application Layout

Converting existing apps to Catalyst

Create A Base Model Class

```
package REISDB;
use base 'DBIx::Class::Schema';

__PACKAGE__->load_classes({
  REISDB => [ qw/InterestRate/ ],
});

1;
```

Application Layout

Converting existing apps to Catalyst

C) Tell Catalyst How to Get Your Data

Using a Catalyst Model Helper

```
$ script/reis_create.pl model REISDB DBIC::Schema REISDB \
dbi:dbdriver:database=dbname;host=db.example.com " " '{AutoCommit=>1}'
```

Stores connection parameters

in ./lib/REIS/Model/REISDB.pm

Associates itself with the base model

./lib/REISDB.pm

Application Layout

Converting existing apps to Catalyst

2. List all actions your app needs to support

We need to be able to support CRUD for the beginning of a usable application

Application Layout

Converting existing apps to Catalyst

2. List all actions your app needs to support

    We need to be able to support CRUD for the beginning of a usable application

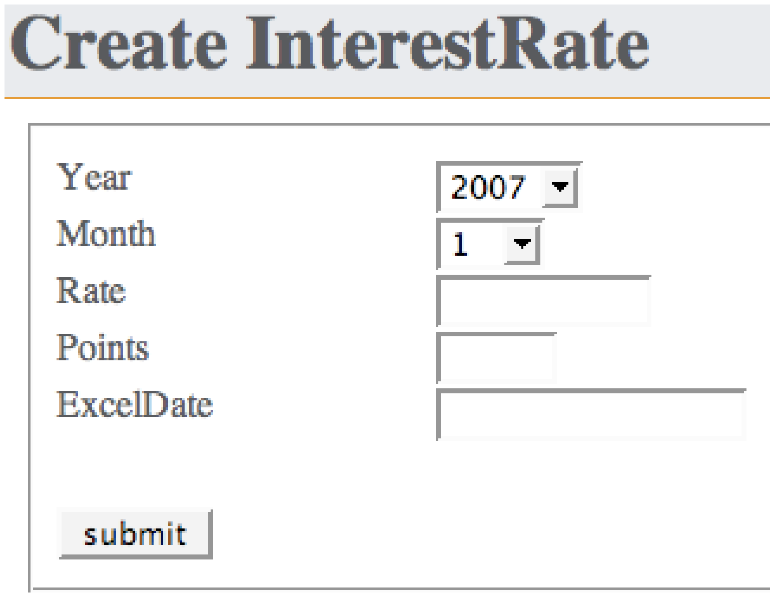
3. Map those actions to descriptive URLs

```
/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete
```

Application Layout

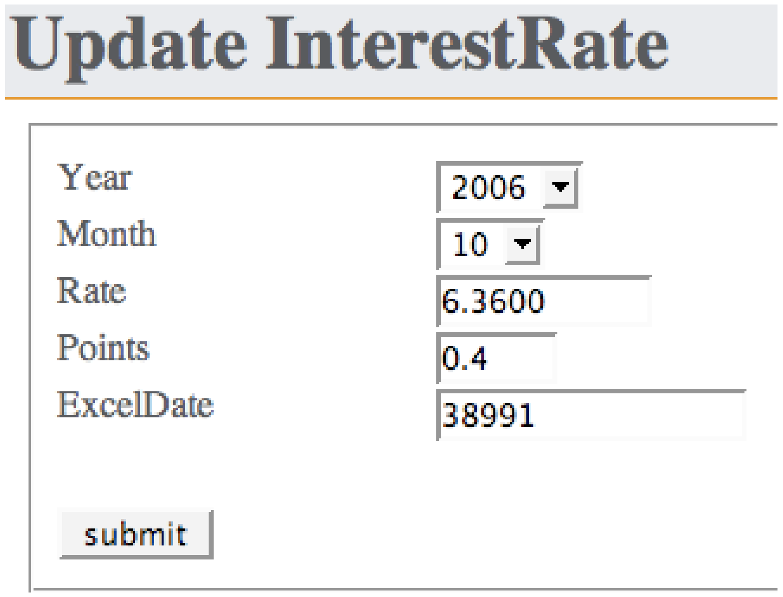
Converting existing apps to Catalyst

4. Design forms to input and update your data



Create InterestRate

Year	2007
Month	1
Rate	
Points	
ExcelDate	



Update InterestRate

Year	2006
Month	10
Rate	6.3600
Points	0.4
ExcelDate	38991

Application Layout

Converting existing apps to Catalyst

The implementation of steps 3 and 4 is interesting, so let's examine those in greater detail

Application Layout
Converting existing apps to Catalyst

The implementation of steps 3 and 4 is interesting, so let's examine those in greater detail

First, use Catalyst's helper to create a controller

```
$ script/reis_create.pl controller InterestRate
... creates a couple files
```

Mapping URLs to Actions

Using Catalyst::Dispatch::Chained

Using Chained Actions gives us an expressive and orderly way to attach Controller methods to URLs

Mapping URLs to Actions

Using Catalyst::Dispatch::Chained

Using Chained Actions gives us an expressive and orderly way to attach Controller methods to URLs

Chained(path) $\longrightarrow$ What needs to run before this action?

Mapping URLs to Actions

Using Catalyst::Dispatch::Chained

Using Chained Actions gives us an expressive and orderly way to attach Controller methods to URLs

Chained(path) $\longrightarrow$ What needs to run before this action?

PathPart(path) $\longrightarrow$ What URL part activates this action?

Mapping URLs to Actions

Using Catalyst::Dispatch::Chained

Using Chained Actions gives us an expressive and orderly way to attach Controller methods to URLs

$$\text{Chained(path)} \longrightarrow \text{What needs to run before this action?}$$

$$\text{PathPart(path)} \longrightarrow \text{What URL part activates this action?}$$

$$\text{Args(int)} \longrightarrow \text{This is an endpoint. URL can contain args.}$$

Mapping URLs to Actions
Using Catalyst::Dispatch::Chained

Using Chained Actions gives us an expressive and orderly way to attach Controller methods to URLs

Chained(path) ⟶	What needs to run before this action?
PathPart(path) ⟶	What URL part activates this action?
Args(int) ⟶	This is an endpoint. URL can contain args.
CaptureArgs(int) ⟶	Not an endpoint. URL can contain args.

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support:

```
/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub list_all : Path Args(0) {
    my ($self, $c) = @_;
    # stash data for display in template
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub list_all : Chained('/') PathPart('interestrates') Args(0) {
    my ($self, $c) = @_;
    # stash data for display in template
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub create : Local {
    my ($self, $c) = @_;
    # build and display data entry form
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub create : Chained('/') PathPart('interestrates/create') Args(0) {
    my ($self, $c) = @_;
    # build and display data entry form
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub view : Chained('/') PathPart('interestrates') Args(1) {
    my ($self, $c, $id) = @_;
    # pull data for one record for display in template
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub _get_id : Chained('/') PathPart('interestrates') CaptureArgs(1) {
    my ($self, $c, $id) = @_;
    $c->stash(ir_id, $id); # stash id to use in actions later in the chain
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub edit : Chained('_get_id') PathPart('edit') Args(0) {
    my ($self, $c) = @_;
    my $id = $c->stash->{ir_id};
    # build edit form for stashed id
}
```

Defining Chained Actions

3. Map actions to descriptive URLs

URL formats to support in our Controller methods:

/interestrates/
/interestrates/create
/interestrates/42
/interestrates/42/edit
/interestrates/42/delete

```
sub delete : Chained('_get_id') PathPart('edit') Args(0) {
    my ($self, $c) = @_;
    my $id = $c->stash->{ir_id};
    # confirm delete and then trash
}
```

Designing Forms

4. Using HTML::Widget for form interaction

```
sub make_ir_widget {
  my ($self, $c) = @_;
  my $w = $c->widget('ir_form')->method('post');

  # get our valid data
  my $thisyear = (localtime(time))[5]+1900;
  my @years = map { $_ => $_ } reverse 1970 .. $thisyear;
  my @months = map { $_ => $_ } 1 .. 12;

  # build the form
  $w->element('Hidden', 'InterestRateID');
  $w->element('Select', 'Year')->label('Year')->options(@years);
  $w->element('Select', 'Month')->label('Month')->options(@months);
  $w->element('Textfield', 'Rate')->label('Rate')->size(10);
  $w->element('Textfield', 'Points')->label('Points')->size(5);
  $w->element('Textfield', 'ExcelDate')->label('ExcelDate')->size(15);
  $w->element('Submit',    'submit')->value('submit');

  return $w;
}
```

Validating Form Input

4. Using HTML::Widget for form interaction

```
# set required fields
  $w->constraint(All => qw/Year Month Rate Points/)->message('Required.');
# must be an integer
  $w->constraint(Integer => 'InterestRateID')->message('Invalid InterestRateID.');
  $w->constraint(Integer => 'ExcelDate')->message('Must be an integer.');
# must be a number within a specified range
  $w->constraint(Range => 'Year')->min(1970)->max($thisyear)
    ->message("Must be in the range 1970-$thisyear.");
  $w->constraint(Range => 'Month')->min(1)->max(12)
    ->message('Must be in the range 1-12.');
# must be a number (optional decimal point, etc.)
  $w->constraint(Number => 'Rate')->message('Must be a number.');
  $w->constraint(Number => 'Points')->message('Must be a number.');
```

Many other types of validation are built in...

Validating Form Input

4. Using HTML::Widget for form interaction

Several Types of Validation Functions

Presence/Dependency

All
AllOrNone
Any
DependOn

Comparison Tests

Equal
In

Predefined Patterns

ASCII
Bool
Date
DateTime
Email
HTTP
Integer
Number
Printable
Range
String
Time

User-Defined Logic

Callback
CallbackOnce
Regex

Very Specific

Length

Lots of Good Docs

They are out there, read them!

<http://search.cpan.org/perldoc?A::Module>

<http://www.catalystframework.org/>

Catalyst::Manual::Tutorial
DBIx::Class::Manual
Template::Toolkit
HTML::Widget
Catalyst::Dispatch::Chained

Speaker Bio

Who is this guy?

Web Programmer with Perl focus since 1999

$dayjob projects revolve around email: ECM/MTA software

We're hiring! <jobs@omniti.com>
<http://www.omniti.com/people/jobs/>

Catalyst helps me be more efficient

with my free time!