Controller Concepts

lesson #basic02

James L. Parry
B.C. Institute of Technology

Agenda

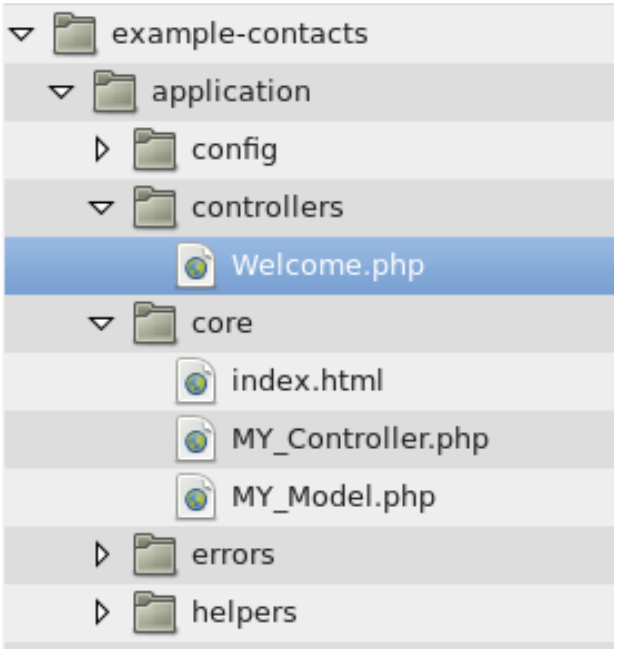
1. Overview
2. Controller Basics
3. Getting Input
4. Using Stuff
5. Producing Output
6. Routing
7. Hooks
8. Loose Ends

Overview

Controllers are a key component of the MVC design pattern.

CodeIgniter controllers are found in the `application/controllers` folder.

If your webapp is extending the built-in controller with your own, it will be found in `application/core/MY_Controller.php`, as you saw in the example-contacts project in lesson 2.

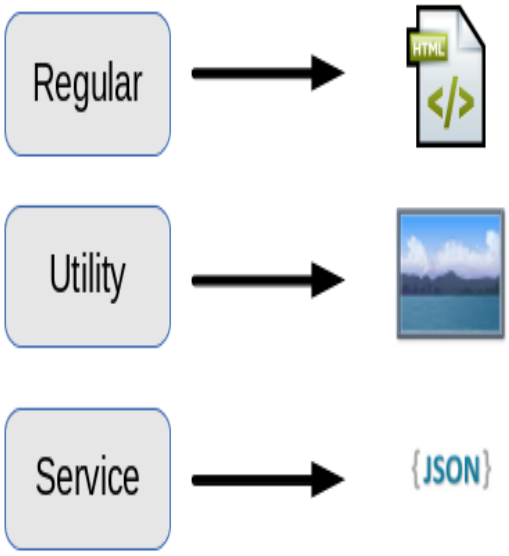


Types of Controllers

There are three common types of controllers:

- A **normal** controller is expected to return a webpage to the browser.
- A **utility** controller is expected to return a file or perhaps a specially formatted data structure.
- A **service** controller is expected to implement the server side of some distributed system protocol (eg. REST).

This week's tutorial will address the first two types, while service controllers will be addressed in the last third of the course.



Controller Basics

Controllers are found in the `application/controllers` folder

Controllers can also be in nested subfolders underneath that.

There is a default controller, `Welcome`, in any folder. This default can be changed, if you like. See the User Guide pages on Controllers and on Routing.

A controller is referenced by its lower-case name. The URL `comp4711.local/product` is a reference to the `Product` controller.

If that same controller were in a subfolder, `application/controllers/inventory`, then it would referenced in a URL as `comp4711.local/inventory/product`.

If "orders" was a subfolder, then the URL `comp4711.local/orders` would be a reference to the `application/orders/Welcome` controller.

Segment Based Naming

URLs are segment based.

For instance, the URL `example.com/class/function/parm1/parm2` is meant to be interpreted as a reference to the `function` method inside the `class` controller, with two positional parameters, `parm1` and `parm2`, passed to the method invocation.

You can't normally pass parameters to a controller's `index` method. It is *possible* to do so, with some clever routing rules, but that is beyond our scope.

You need to be careful with your selection of controller class and method names, or the handling of a request by CodeIgniter might not be what you expect!

For example, the URL `/apple/banana/pear` *could* be resolved by any of the following:

- `Apple::banana(pear)` or
- `apple/Banana::pear()` or
- `apple/banana/Pear::index()` or
- `apple/banana/pear/Welcome::index()`

Getting Input

Your controller normally gets data passed as positional parameters in the URL.

For instance, if your method looks like...
`function order($item,$quantity) {...}`
then the URL
`.../order/apple/1`
will result in `$item` having the value `"apple"` and `$quantity` having the value `1` inside your method.

Getting Input From a Form

If you are processing an HTML form, then the form fields are accessed inside your handling method by using the Input class.

Retrieve field values from the submission using
`$this->input->post(fieldname)`.

For instance,
`$item_code = $this->input->post('item');`
will assign the submitted value of the form field with the name attribute "item" to the local variable `$item_code`.

Note that fields are referenced according to their "name" attribute, and not their "id" attribute.

Getting Input Other Ways

It is *possible* to enable query strings, so that input could be passed in the URL as query parameters (`controller?a=...&b=...`), but strongly discouraged.

It is also possible to access the PHP superglobals (`$_POST` or `$_GET`) directly, but you are then responsible for checking for paremeter existence, and doing your own data cleansing on the values.

CodeIgniter has a File uploading class . We will address this topic in lesson 5

Using Stuff

"Stuff" is a highly technical term, in this context meaning any support component that comes with CodeIgniter or that you created.

Stuff includes models, libraries, helpers, drivers, and configuration settings.

A slide for each of these follows, with a simple introduction. Each of the kinds of support component is addressed, in more detail, in upcoming lessons.

Finding Stuff

CodeIgniter has its own loader, which will generally handle a support component `something` by looking for `application/xxx/Something`, and then for `system/xxx/CI_Something`, where "xxx" is the kind of component. This is a simplification of the actual process, which will be described in more detail in later lessons.

A controller property, `$this->something`, is thereafter available inside your controller.

The CodeIgniter loader itself is injected as an object property of your controller, accessible as `$this->load`.

Model Stuff

Model stuff is loaded using the CodeIgniter loader, as in `$this->load->model('products');`

The loader will look for `application/models/Products`, which is expected to extend a class defined in `application/core/MY_Model` or else `system/core/CI_Model`.

A controller property, `$this->products`, is thereafter available inside your controller.

Note: "products" above is an example model name.

Library Stuff

Library stuff is loaded using the CodeIgniter loader, as in `$this->load->library('supporter');`

The loader will look for `application/libraries/Supporter`, and then for `system/libraries/CI_Supporter`.

A controller property, `$this->supporter`, is thereafter available inside your controller.

Note: "supporter" above is an example library name.

Driver Stuff

Drivers are special libraries. Each one has a core driver class, and then adapters specific to the kind of driver, for instance providing an implementation for a particular database engine.

Driver stuff is loaded using the CodeIgniter loader, as in
`$this->load->driver('session');`

The loader will look for `system/libraries/Session/Session`. The actual driver object returned will come from inside `system/libraries/Session/drivers/`, and will be chosen based on driver configuration data that you have specified in `application/config/...`.

A controller property, `$this->session`, is thereafter available inside your controller.

Note: "session" above is used as an example.

Helper Stuff

Helper stuff is loaded using the CodeIgniter loader, as in
`$this->load->helper('useful');`

The loader will look for `application/helpers/useful_helper`, and then for `system/helpers/useful_helper`.

Helpers can be extended too, but that is an advanced topic.

When a helper is loaded, the functions defined inside it are then available to your controller.

Configuration Stuff

CodeIgniter automatically loads its configuration class, and injects it as a `config` property of your controller.

You can retrieve configuration settings by
`$this->config->item(...);`

Your configuration settings are made through key/value pairs in the different files in `application/config`.

Producing Output

Your controller can produce output by loading a view from inside your `application/views` folder.
`$this->load->view(name[,parms[,capture]]);`

This technique simply copies the contents of the named PHP webpage to the output stream.

The name of the view file can include subfolders, and *is* case-sensitive, unlike the names used to reference other "stuff".

Template Parser

CodeIgniter also includes a simple template parser, which can do simple substitutions. It also uses view files from inside your `application/views` folder.
`$this->parser->parse(name[,parms[,capture]]);`

This technique copies the contents of the named PHP webpage to the output stream, after substituting fields specified inside braces (eg `{somefield}`).

The template parser needs to be loaded, as a library, before it can be used.

You saw this earlier, in the contacts example from lesson 2.

Template Parser Example

Here is a simple example of field substitution using the template parser.

Controller:
```
$parms=array('username'=>'Jim');
$this->parser->parse('blah',$parms);
```

View:
```
<p>Hi there, {username}</p>
```

Result:
```
Hi there, Jim
```

Template Parser Parameters

Parameters are passed as associative arrays
```
$parms=array('name'=>'Jim','code'=>'123');
```

Parameters can be nested...
```
$parms=array( 'items' => array(
    array('itemcode'=>'1','desc'=>'burger'),
    array('itemcode'=>'2','desc'=>'fries'),
  ), );
```

These would be used like...
```
{items} <p>{itemcode} means {desc}</p> {/items}
```

And the result for the above would look like...
```
1 means burger
2 means fries
```

Controller, Revisited

So, putting the previous slides together gives the following steps for each controller's handling method:

1. Load any needed components
2. Extract input parameters
3. Access your model(s)
4. Validate and process your input, updating any models
5. Build view parameters array
6. Load the proper response view or template

Routing

The controller folder convention can be over-ridden by specifying routing rules, in `application/config/routes.php`

An example such rule lets you change the default controller:
```
$route['default_controller'] = 'welcome';
```

If you specify multiple rules, they are tested consecutively until one fits.

Wildcard Routing

A routing rule can use a "wildcard" token, `(:num)` to match a numeric segment value, or `(:any)` to match any segment value.

Specify an expression using these as the "key" for a routing rule, and specify the proper destination as the "value".

You can use the substitution token `$n` to reference a URI segment in the original request.

Some examples of routing rules:

```
$route['blog/joe'] =
"blogs/users/34";

$route['product/(:num)'] =
"catalog/product_lookup_by_id/$1";

$route['page/(:any)'] =
'welcome/page/$1';

$route['secret'] =
'youllneverfindme/$1/$2';
```

Regular Expression Routing

You can also use a regular expression in a routing rule.

For instance:
`$route['products/([a-z]+)/(\d+)'] = "$1/id_$2";`
would remap `/products/banana/eat` to `/eat/id_banana`

Another example:
`$route['([a-z]+)/register'] = 'assimilate/$0';`
would remap `/jim/register` to `/assimilate/jim`

Callback Routing

If you are using PHP >= 5.3 you can use callbacks in place of the normal routing rules to process the back-references.

For instance:

```
$route['products/([a-zA-Z]+)/edit/(\d+)'] =
function ($product_type, $id)
{
  return 'catalog/product_edit/' . strtolower($product_type) . '/' .
$id;
};
```

Author's note: this is new to me, and I don't have a good explanation for it (yet).

HTTP Verb Routing

You can specify routing rules that apply to specific HTTP request types. This would be applicable to utility and service controllers.

Some examples, in a RESTful fashion:

```
$route['products']['PUT'] = 'product/insert';
$route['products/(:num)']['DELETE'] = 'product/delete/$1';
```

Hooks

The CodeIgniter framework, internally, performs the following steps to handle a request:

1. Apply routing rules to determine the controller and method to use
2. Instantiate the controller
3. Invoke the appropriate method, capturing output
4. Return the output to the browser

CodeIgniter also provides "hooks", to let you inject processing at various pre-defined stages of the request handling.

Refer to the user guide for details!

Hook Points

The following are some of the "hook points" that you can use:

- pre_system
- pre_controller
- post_controller_constructor
- post_controller
- display_override
- post_system

Hooks are configured similarly to routes, and you can have multiple hooks for the same hook point.

Adding Hooks

Configure your hooks in `application/config/hooks`

An example:

```
$hookie = array(
  'class'=>...,
  'function'=>...,
  'filename'=>...,
  'filepath'=>...,
  'params'=>...
);

$hooks[entrypoint][] = $hookie;
```

Loose Ends

Upcoming lessons will address some more exotic controller issues:

- Webapp error handling (avoid getting fired)
- Handling AJAX requests
- Handling service requests
- Handling plugins for additional resources

Caution: Before using a feature, eg. hooks or routing, RTFM!

Coding Conventions

Required:

- Class and file naming - "ucfirst"

Allowed:

- deviations from the suggested, for good reason
- multiple classes (related) in file

Bad ideas:

- PHP namespaces (for now)

Suggested (for methods & variables):

- words separated by underscores
- underscores in front of internal items
- Allman style braces & indenting
- commenting, Javadoc style!
- value & type comparison (===)!
- don't use closing PHP tag at end of file!

Congratulations!

You have completed lesson #basic02: Controller Concepts

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: basic03 Models

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.