Aura Framework v2

The missing Manual

Hari K T and Paul M. Jones

Aura Framework v2

The missing Manual

Hari K T and Paul M. Jones

This book is for sale at http://leanpub.com/aurav2

This version was published on 2016-11-01

Tweet This Book!

Please help Hari K T and Paul M. Jones by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I bought a copy of Aura Framework v2 : The missing manual, to support authors. You can also read free #aurav2manual

The suggested hashtag for this book is #aurav2manual.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#aurav2manual

Contents

Routing

Configuration of routing and dispatching is done via the project-level config/ class files. If a route needs to be available in every config mode, edit the project-level config/Common.php class file. If it only needs to be available in a specific mode, e.g. dev, then edit the config file for that mode (`config/Dev.php`).

The `modify()` method is where we get the router service (‘aura/web-kernel:router’) and add routes to the application.

```php
<?php
namespace Aura\Framework_Project\_Config;

use Aura\Di\Config;
use Aura\Di\Container;

class Common extends Config
{
    public function define(Container $di)
    {
        // define params, setters, and services here
    }

    public function modify(Container $di)
    {
        // get the router service
        $router = $di->get('aura/web-kernel:router');
        // ... your application routes go below
    }
}
```

The `aura/web-kernel:router` is an object of type *Aura\Router\Router* . So if you are familiar with Aura.Router[1] then you are done with this chapter, else read on.

Aura framework can act both as a micro framework or full stack framework. If you are using it as a micro framework, you can set a Closure as the action value, else set the same name of the action in the dispatcher. Don’t worry, we will cover dispatching in next chapter.

> Note: This chapter gives you a basic understanding of the different types of methods available in router.

[1]https://github.com/auraphp/Aura.Router

Adding a Route

We will not be showing the whole config file to reduce the space used. This document assumes you are adding the route in the `modify()` method after getting the router service.

To create a route, call the `add()` method on the *Router*. Named path-info params are placed inside braces in the path.

```php
// add a simple named route without params
$router->add('home', '/');

// add a simple unnamed route with params
$router->add(null, '/{action}/{id}');

// add a named route with an extended specification
$router->add('blog.read', '/blog/read/{id}{format}')
    ->addTokens(array(
        'id'     => '\d+',
        'format' => '(\.[^/]+)?',
    ))
    ->addValues(array(
        'action'     => 'BlogReadAction',
        'format'     => '.html',
    ));
```

You can create a route that matches only against a particular HTTP method as well. The following *Router* methods are identical to `add()` but require the related HTTP method:

- `$router->addGet()`
- `$router->addDelete()`
- `$router->addOption()`
- `$router->addPatch()`
- `$router->addPost()`
- `$router->addPut()`

Advanced Usage

Extended Route Specification

You can extend a route specification with the following methods:

- `addTokens()` – Adds regular expression subpatterns that params must match.

```
addTokens(array(
    'id' => '\d+',
))
```

Note that `setTokens()` is also available, but this will replace any previous subpatterns entirely, instead of merging with the existing subpatterns.

- `addServer()` – Adds regular expressions that server values must match.

```
addServer(array(
    'REQUEST_METHOD' => 'PUT|PATCH',
))
```

Note that `setServer()` is also available, but this will replace any previous expressions entirely, instead of merging with the existing expressions.

- `addValues()` – Adds default values for the params.

```
addValues(array(
    'year' => '1979',
    'month' => '11',
    'day' => '07'
))
```

Note that `setValues()` is also available, but this will replace any previous default values entirely, instead of merging with the existing default value.

- `setSecure()` – When `true` the `$server['HTTPS']` value must be on, or the request must be on port 443; when `false`, neither of those must be in place.
- `setWildcard()` – Sets the name of a wildcard param; this is where arbitrary slash-separated values appearing after the route path will be stored.
- `setRoutable()` – When `false` the route will be used only for generating paths, not for matching (`true` by default).
- `setIsMatchCallable()` – A custom callable with the signature `function(array $server, \ArrayObject $matches)` that returns true on a match, or false if not. This allows developers to build any kind of matching logic for the route, and to change the `$matches` for param values from the path.
- `setGenerateCallable()` – A custom callable with the signature `function(\ArrayObject $data)`. This allows developers to modify the data for path interpolation.

Here is a full extended route specification named `read`:

```php
$router->add('blog.read', '/blog/read/{id}{format}')
    ->addTokens(array(
        'id' => '\d+',
        'format' => '(\.[^/]+)?',
        'REQUEST_METHOD' => 'GET|POST',
    ))
    ->addValues(array(
        'id' => 1,
        'format' => '.html',
    ))
    ->setSecure(false)
    ->setRoutable(false)
    ->setIsMatchCallable(function(array $server, \ArrayObject $matches) {

        // disallow matching if referred from example.com
        if ($server['HTTP_REFERER'] == 'http://example.com') {
            return false;
        }

        // add the referer from $server to the match values
        $matches['referer'] = $server['HTTP_REFERER'];
        return true;

    })
    ->setGenerateCallable(function (\ArrayObject $data) {
        $data['foo'] = 'bar';
    });
```

Default Route Specifications

You can set the default route specifications with the following *Router* methods; the values will apply to all routes added thereafter.

```php
// add to the default 'tokens' expressions; setTokens() is also available
$router->addTokens(array(
    'id' => '\d+',
));

// add to the default 'server' expressions; setServer() is also available
$router->addServer(array(
    'REQUEST_METHOD' => 'PUT|PATCH',
));

// add to the default param values; setValues() is also available
$router->addValues(array(
    'format' => null,
));

// set the default 'secure' value
$router->setSecure(true);

// set the default wildcard param name
$router->setWildcard('other');

// set the default 'routable' flag
$router->setRoutable(false);

// set the default 'isMatch()' callable
$router->setIsMatchCallable(function (...) { ... });

// set the default 'generate()' callable
$router->setGenerateCallable(function (...) { ... });
```

Simple Routes

You don’t need to specify an extended route specification. With the following simple route …

```php
$router->add('archive', '/archive/{year}/{month}/{day}');
```

… the *Router* will use a default subpattern that matches everything except slashes for the path params. Thus, the above simple route is equivalent to the following extended route:

```php
$router->add('archive', '/archive/{year}/{month}/{day}')
    ->addTokens(array(
        'year'  => '[^/]+',
        'month' => '[^/]+',
        'day'   => '[^/]+',
    ));
```

Automatic Params

The *Router* will automatically populate values for the `action` route param if one is not set manually.

```php
// ['action' => 'foo.bar'] because it has not been set otherwise
$router->add('foo.bar', '/path/to/bar');

// ['action' => 'zim'] because we add it explicitly
$router->add('foo.dib', '/path/to/dib')
       ->addValues(array('action' => 'zim'));

// the 'action' param here will be whatever the path value for {action} is
$router->add('/path/to/{action}');
```

Optional Params

Sometimes it is useful to have a route with optional named params. None, some, or all of the optional params may be present, and the route will still match.

To specify optional params, use the notation `{/param1,param2,param3}` in the path. For example:

```php
$router->add('archive', '/archive{/year,month,day}')
    ->addTokens(array(
        'year'  => '\d{4}',
        'month' => '\d{2}',
        'day'   => '\d{2}'
    ));
```

The leading slash separator is inside the params token, not outside.

With that, the following routes will all match the 'archive' route, and will set the appropriate values:

```
/archive
/archive/1979
/archive/1979/11
/archive/1979/11/07
```

Optional params are *sequentially* optional. This means that, in the above example, you cannot have a "day" without a "month", and you cannot have a "month" without a "year".

Only one set of optional params per path is recognized by the *Router*.

Optional params belong at the end of a route path; placing them elsewhere may result in unexpected behavior.

Wildcard Params

Sometimes it is useful to allow the trailing part of the path be anything at all. To allow arbitrary trailing params on a route, extend the route definition with `setWildcard()` to specify param name under which the arbitrary trailing param values will be stored.

```
$router->add('wild_post', '/post/{id}')
    ->setWildcard('other');
```

Attaching Route Groups

You can add a series of routes all at once under a single "mount point" in your application. For example, if you want all your blog-related routes to be mounted at `/blog` in your application, you can do this:

```
$name_prefix = 'blog';
$path_prefix = '/blog';

$router->attach($name_prefix, $path_prefix, function ($router) {

    $router->add('browse', '{format}')
        ->addTokens(array(
            'format' => '(\.json|\.atom|\.html)?'
        ))
        ->addValues(array(
            'format' => '.html',
        ));

    $router->add('read', '/{id}{format}', array(
        ->addTokens(array(
            'id'     => '\d+',
            'format' => '(\.json|\.atom|\.html)?'
        )),
        ->addValues(array(
            'format' => '.html',
        ));

    $router->add('edit', '/{id}/edit{format}', array(
        ->addTokens(array(
            'id' => '\d+',
            'format' => '(\.json|\.atom|\.html)?'
        ))
        ->addValues(array(
            'format' => '.html',
        ));
});
```

Each of the route names will be prefixed with 'blog.', and each of the route paths will be prefixed with `/blog`, so the effective route names and paths become:

- `blog.browse => /blog{format}`
- `blog.read => /blog/{id}{format}`
- `blog.edit => /blog/{id}/edit{format}`

You can set other route specification values as part of the attachment specification; these will be used as the defaults for each attached route, so you don't need to repeat common information. (Setting these values will not affect routes outside the attached group.)

```
$name_prefix = 'blog';
$path_prefix = '/blog';

$router->attach($name_prefix, $path_prefix, function ($router) {

    $router->setTokens(array(
        'id'     => '\d+',
        'format' => '(\.json|\.atom)?'
    ));

    $router->setValues(array(
        'format' => '.html',
    ));

    $router->add('browse', '');
    $router->add('read', '/{id}{format}');
    $router->add('edit', '/{id}/edit');
});
```

Attaching REST Resource Routes

The router can attach a series of REST resource routes for you with the `attachResource()` method:

```
$router->attachResource('blog', '/blog');
```

That method call will result in the following routes being added:

Route Name	HTTP Method	Route Path	Purpose
blog.browse	GET	/blog{format}	Browse multiple resources
blog.read	GET	/blog/{id}{format}	Read a single resource
blog.edit	GET	/blog/{id}/edit	The form for editing a resource
blog.add	GET	/blog/add	The form for adding a resource
blog.delete	DELETE	/blog/{id}	Delete a single resource
blog.create	POST	/blog	Create a new resource
blog.update	PATCH	/blog/{id}	Update part of an existing resource
blog.replace	PUT	/blog/{id}	Replace an entire existing resource

The `{id}` token is whatever has already been defined in the router; if not already defined, it will be any series of numeric digits. Likewise, the `{format}` token is whatever has already been defined in the router; if not already defined, it is an optional dot-format file extension (including the dot itself).

The `action` value is the same as the route name.

If you want calls to `attachResource()` to create a different series of REST routes, use the `setResourceCallable()` method to set your own callable to create them.

```php
$router->setResourceCallable(function ($router) {
    $router->setTokens(array(
        'id' => '([a-f0-9]+)'
    ));
    $router->addPost('create', '/{id}');
    $router->addGet('read', '/{id}');
    $router->addPatch('update', '/{id}');
    $router->addDelete('delete', '/{id}');
});
```

The example will cause only four CRUD routes, using hexadecimal resource IDs, to be added for the resource when you call `attachResource()`.