# aurelia基础知识--router

## Basic Configuration

为了使用aurelia的路由，你的组件视图需要包含一个标签<router-view></router-view>。为了配置路由，组件的view-model需要一个configureRouter()方法。

<template>

<ul repeat.for="nav of router.navigation">

<li class="${nav.isActive ? 'active' : ''}"><a href.bind="nav.href">${nav.title}</a></li>

</ul>

<router-view></router-view>

</template>

export class App {

configureRouter(config, router) {

this.router = router;

config.title = 'Aurelia';

config.map([

{ route: ['', 'home'], name: 'home', moduleId: 'home/index' },

{ route: 'users', name: 'users', moduleId: 'users/index', nav: true, title: 'Users' },

{ route: 'users/:id/detail', name: 'userDetail', moduleId: 'users/detail' },

{ route: 'files/\*path', name: 'files', moduleId: 'files/index', nav: 0, title: 'Files', href:'#files' }

]);

}

}

router.isNavigating：如果路由真处于一个导航中，那么该值为True，记住，他只有在你出于基础路由而不是子路由中起作用。

config.map()：一个路由的容器。不仅有上面的例子中route，name，moduleId，href，nav这些参数，还有一些其他参数可以包含在路由中。路由的接口名是RouteConfig。你可以使用config.mapRoute()去添加一个单独的路由。

route：它是匹配传入的URL片段的模式。它可以是字符串或者一个字符串数组。该路由还可以包含参数化路由或通配符

* + 参数化路由与字符串:token匹配。例如'users/:id/detail'。路由将会设置一个token参数名的对象，并把它作为一个配置传递给路由的view-model的activate()方法。
  + 可以通过附加一个“?”来选择一个参数。例如，'users/:id?/detail'可以匹配users/3/detail以及users/detail。当url中缺失配置参数，马儿会传递给activate()一个undefined的参数。
  + 通配符被用作匹配类似于“rest”的路径。例如files/\*path matches files/new/doc or files/temp。一个带有URL被分割后的剩余部分的对象被设置为path属性作为参数传递给activate()。

Href：是有条件的可选属性。如果没有定义，则使用路由。如果路由是分段的，那么href就需要在文件的情况下，因为路由器不知道如何填充模式的参数化部分。

Nav：当设置为true时，路由将包含在路由器的导航模型中。如果是一个number，其值江湖被用作路由的排序。这使得创建动态菜单或类似元素变得更加容易。导航模型将在router.navigation中作为一个NavModel的数组而可用。一下是NavModel可用的参数：

* + isActive当他关联的路由是激活状态是，值为true
  + title当他关联的路由被激活时，将会附加到html的title上。
  + href可以被使用在a标签上。
  + config定义在config.map上的对象。
  + settings与config对象的settings属性等同。
  + router一个AppRouter的引用

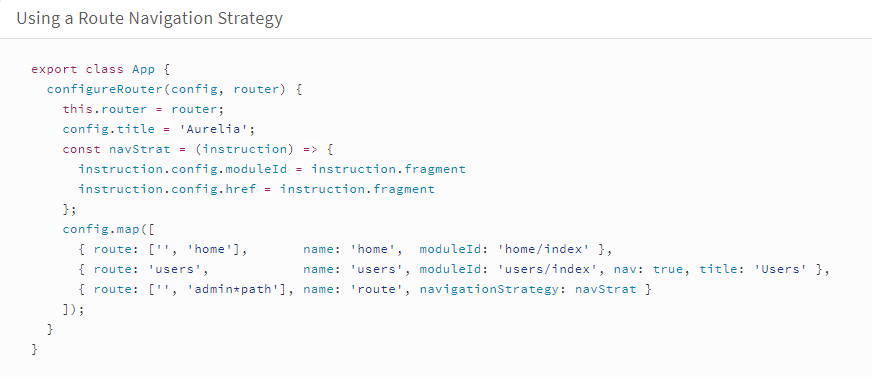
其他的配置还有relativeHref和order

## Options

没什么好说的，用处不大，就不翻译了。

## Dynamically Specify Route Components

你可以添加一个导航策略到路由中，从而允许动态路由。在导航策略中aurelia需要你使用moduleId，viewPorts或者redirect去配置instructin.config。



## Adding Additional Data To A Route

尽管Aurelia确实允许你将任何额外的属性传递给路由的配置对象，但settings是默认的参数，你应该添加你想要传递到路由的任意数据。

export class App {

configureRouter(config, router) {

this.router = router;

config.title = 'Aurelia';

config.map([

{ route: ['', 'home'], name: 'home', moduleId: 'home/index' },

{ route: 'users', name: 'users', moduleId: 'users/index', nav: true, title: 'Users', settings: {data: '...'} }

]);

}

}

## Case Sensitive Routes

你可以通过设置caseSensitive: true使路由大小写敏感。

export class App {

configureRouter(config, router) {

this.router = router;

config.title = 'Aurelia';

config.map([

{ route: ['', 'home'], name: 'home', moduleId: 'home/index' },

{ route: 'users', name: 'users', moduleId: 'users/index', nav: true, title: 'Users', caseSensitive: true }

]);

}

}

## Handling Unknown Routes

aurelia允许你使用mapUnknownRoutes()设置一个位置路由，用于当路由没有匹配到时，传递给他的参数可以是：

一个moduleId字符串

一个routeConfig对象

一个函数





## Redirecting Routes

Aurelia允许通过指定一个由URL片段组成的字符串重定向到URL片段。

config.map([

{ route: '', redirect: 'home' },

{ route: 'home', name: 'home', moduleId: 'home/index' }

]);

当你有一个空route片段的时候，例如例子中的home，重定向是非常有用的。在这种情况下，创建一个非空路由，然后将空路由重定向到非空路由(如上所示)这将使子路由器能够始终匹配子路由，而不会在空路由匹配的情况下混淆

## Pipelines

aurelia有两个路由类，AppRuter和Router。AppRouter继承Router，是应用的主要路由。Router被应用于任何子路由器，包括嵌套的子路由器。

默认的pipelines插槽顺序是authorize，preActivate，preRender，postRender。

对于每个插槽aurelia都有便利的方法为其提供pipeline步骤。 addAuthorizeStep：如果定义了，会在载入路由步骤和调用路由的viewmodel的canActivate函数之间被调用。

addPreActivateStep：如果定义了，会在canActivate和deactivate之间被调用。

addPreRenderStep：如果定义了，会在activate函数和组件被渲染或构成前调用。

addPostRenderStep：如果定义了，在组件被渲染或构成后调用。

一个pipeline step必须是一个对象，并且包含一个run(navigationInstruction, next)方法。

例如如下是个登录验证的实现。

import {Redirect} from 'aurelia-router';

export class App {

configureRouter(config) {

const step = new AuthorizeStep;

config.addAuthorizeStep(step);

config.map([

{ route: ['', 'home'], name: 'home', moduleId: 'home/index' },

{ route: 'users', name: 'users', moduleId: 'users/index', settings: { auth: true } },

{ route: 'users/:id/detail', name: 'userDetail', moduleId: 'users/detail', settings: { auth: true } }

]);

}

}

class AuthorizeStep {

run(navigationInstruction, next) {

if (navigationInstruction.getAllInstructions().some(i => i.config.settings.auth)) {

var isLoggedIn = // insert magic here;

if (!isLoggedIn) {

return next.cancel(new Redirect('login'));

}

}

return next();

}

}

## Rendering View Ports

每一个router-view自定义元素的实例都会定义一个“view port”。当你为router-view赋予一个名字，你可以路由配置的viewPorts属性中引用它。viewPorts属性的值是一个对象，它的每个属性都是一个view port的名称，每个值都是指向目的路由的moduleId。因此，你可以在路由配置中指定任意多个view port。如果你不定义router-view的名字，他会以‘default’名字出现。

例如

<template>

<div>

<router-view name="left"></router-view>

</div>

<div>

<router-view name="right"></router-view>

</div>

</template>

export class App {

configureRouter(config) {

config.title = 'Aurelia';

config.map([

{ route: 'users', name: 'users', viewPorts: { left: { moduleId: 'user/list' }, right: { moduleId: 'user/detail' } } }

]);

}

}

## Layouts

类似于MVC风格的主/布局风格页面。aurelia允许你使用一个‘layout’视图类似于MVC的主模板去设置视图。

从技术上讲，路由引用的是‘moduleIds’，而不是‘views’。因为路由器将一个moduleId解析为一个view，间接地，路由器会引用一个视图。在一个布局中绘制一个view是很容易的，因此在这个主题中，我们通过路由引用视图，而不是模块。

我们先来看一下HTML。我们知道，roter-view自定义HTML元素总是与在其父视图的视图模型中给定的路由器配置中引用的一组或多个视图相关联。通过使用router-view关联一个布局，我们因此可以将一个布局和这个router-view关联的同一组视图关联起来。

在router-view自定义元素中指定一个布局，我们可以使用如下的参数：

* layout-view 指定布局视图的文件名（带路径）。
* layout-view-model 指定布局视图的view model的moduleId。
* layout-model 指定传递给布局的view model中active方法的模块参数。

## View Swapping and Animation

当aurelia路由从一个导航到另一个的时候，我们也可以成为从一个视图交换到另一个视图。aurelia提供给我们了一些配置设置的策略命令去指定如何交换，或者更明确地说，交换动画如何展示。我们将这些策略更多地成为‘swap order’。如果没有设置动画。那么swap-order将不会有视觉影响。你可以通过设置‘swap-order’参数到router-view自定义html元素中，从而将交换策略应用到一个或多个路由中。策略就会被应用到与router-view关联的任意两个视图的切换中。‘swap-order’是可以绑定的。

下面的swap order策略是可用的：

* before - 在删除当前视图前激活下一个视图。
* with - 在删除当前视图的同时激活下一个视图。
* after - 在删除当前视图后激活下一个视图（默认）。

例如

<template>

<div>

<router-view swap-order="before"></router-view>

</div>

</template>

## Internationalizing Titles

如果你的应用支持多种语言，你也许会想翻译你的路由标题。Router类有一个transformTitle属性可以使用。他将会被分配一个函数，从而将当前活动的路由的标题作为一个参数并返回翻译后的标题。

## Configuring a Fallback Route

在有时，如果导航被拒绝，他会重定向到上一个位置，如果上一个位置不存在，为了应对这种情况，你可以设置一个fallback路由。

export class App {

configureRouter(config, router) {

this.router = router;

config.title = 'Aurelia';

config.map([

{ route: ['', 'home'], name: 'home', moduleId: 'home/index' },

{ route: 'users', name: 'users', moduleId: 'users/index', nav: true, title: 'Users' }

]);

config.fallbackRoute('users');

}

}

## Reusing an Existing View Model

由于view model的声明周期只被调用一次，你可以能遇到用户将路由从ProductA切换到ProductB的问题。为了解决这个问题，你需要实现在你的view model中determineActivationStrategy这个方法，并返回路由关于你希望发生什么的提示。可用的返回值又replace和invoke-lifecycle。记住，声明周期引用的是导航的生命周期。

export class App {

configureRouter(config) {

config.title = 'Aurelia';

config.map([

{ route: 'product/a', moduleId: 'product', nav: true },

{ route: 'product/b', moduleId: 'product', nav: true },

]);

}

}

//product.js

import {activationStrategy} from 'aurelia-router';

export class Product {

determineActivationStrategy() {

return activationStrategy.replace;

}

}

如果策略总是一样的，并且你不希望将他们写在你的view model代码中。你可以添加在你的路由设置中添加*activationStrategy来替代。*