Aurelia框架基础知识——html行为

# HTML行为

Aurelia中的术语“HTML行为”包含两个基本概念：自定义元素和自定义属性。

自定义元素：顾名思义，是你的标记的元素。通常会有一些标记，这些标记将作为应用程序页面的一部分呈现。另一方面，“自定义属性”可用于在标准HTML元素中（如div或button）或“自定义元素”中以更改元素的行为。自定义元素和自定义属性都支持Aurelia强大的数据绑定。

HTML行为要求您为行为创建一个类。这个类对于自定义元素和自定义属性都被称为“视图模型”。

Aurelia命名约定：自定义元素的视图模型通常与HTML“View”配对。在你的视图模型的名称后追加CustomElement或CustomAttribute来告诉框架某个类是自定义元素或自定义属性的视图模型。

题外知识：JavaScript类的标准命名约定是InitCaps，意思是每个字母首字母大写; 然而，HTML是不区分大小写的，通常归一化为小写。为了缩小这个差距，Aurelia将采取类名称，去掉它CustomElement或CustomAttribute后缀，然后将类名转换为dash-case。这意味着，当你在模板中使用它时，HelloWorldCustomElement要变为hello-world和RedSquareCustomAttribute要变为red-square。

当然Aurelia也提供了覆盖命名约定的方式：通过在你的HTML行为的视图模型中使用@customAttribute或@customElement装饰器来明确命名你的HTML行为。你传递一个字符串到装饰器，这成为你的行为将使用的名称。Aurelia不会以任何方式改变这个字符串。这意味着@customElement('helloworld')会给你一个名为<helloworld>的自定义元素。这两个装饰器可以从aurelia-framework模块导入。在使用这些装饰器之一时，不需要遵循上面提到的命名约定，因为装饰器的使用告诉Aurelia您正在创建自定义元素或自定义属性。

例如：

import {customAttribute} from 'aurelia-framework';

@customAttribute('red-square')

export class BlueSquare {

}

当告诉Aurelia将Element类的实例注入到类中时，Aurelia将注入与HTML行为关联的DOM元素。

例如：

import {inject} from 'aurelia-framework';

@inject(Element)

export class RedSquareCustomAttribute {

constructor(element){

this.element = element;

this.element.style.width = this.element.style.height = '100px';

this.element.backgroundColor = 'red';

}

}

在上面的例子中，我们使用了浏览器全局：Element。有时使用浏览器全局可能会导致测试问题。情况并非如此。但是，如果您希望利用服务器端渲染，则需要使代码库免受浏览器全局性影响。你可能要提前做好准备。为了帮助您避免浏览器全局，Aurelia提供了一个平台抽象层（PAL），其提供了三个出口DOM，FEATURE和PLATFORM。在上述情况下，你可以用DOM.Element代替Element。

以上只是说明了如何创建HTML行为，那么如何使用它们呢。首先我们必须在视图中提供它。HTML行为（和其他视图资源）默认情况下是全局不可用。这意味着您必须让Aurelia知道在每个视图中可用的资源。这是使用require元素完成的。该require元素有两个属性from和as。您必须为该from属性提供一个值，但是as是可选的。您不能在require上使用数据绑定from或as属性。当“需要”一个视图资源与require元素，Aurelia将使其可用于你的视图。

例如：

<template>

<require from="my-custom-element"></require>

<my-custom-element></my-custom-element>

</template>

from属性中提供的路径可以采用以下两种形式之一：它可以相对于应用程序的根或相对于当前视图的路径。没有./或../启动路径的路径将是相对于你的应用程序的根，而一个路径./或../将是相对于你的视图的路径。

例如

<template>

<require from="./my-custom-element-one"></require>

<require from="../another-dir/my-custom-attribute"></require>

<my-custom-element-one></my-custom-element-one>

<div my-custom-attribute></div>

</template>

资源在您的视图中所采用的名称可以通过以下三种方式之一来确定：

1. 按照Aurelia惯例，
2. 通过资源中的明确命名，
3. 使用as属性覆盖。

如果碰巧使用具有相同名称的多个资源，则必须用as属性为所有冲突资源提供一个值。只要您想要为资源起个别名，也可以使用该属性。也许HTML行为的标准名称很长，你想给它一个简短的名字。该as属性可以帮助你。

例如：

<template>

<require from="my-custom-element-one"></require>

<require from="../another-dir/my-custom-element-one" as="override-the-name"></require>

<my-custom-element-one></my-custom-element-one>

<override-the-name></override-the-name>

</template>

通常不需要为from属性添加文件扩展名。首先，开发环境中文件的文件扩展名可能与浏览器运行的文件扩展名不同（例如.ts使用TypeScript时的文件）。其次，大多数自定义元素都需要Aurelia同时引入javascript 视图模型和HTML 视图。Aurelia的加载器将决定哪个文件扩展名需要附加到您提供的文件名。这个规则有一个例外就是Aurelia提供了“纯HTML的自定义元素”。您必须通过在from属性提供.html扩展名来告诉Aurelia您正在使用一个只有HTML的自定义元素。

例如：

<template>

<require from="my-custom-element"></require>

<require from="./my-html-only-custom-element.html"></require>

<my-custom-element></my-custom-element>

<my-html-only-custom-element></my-html-only-custom-element>

</template>

需要复用的某些资源可以通过FrameworkConfiguration类提供的globalResources，把一个或多个的资源路径的字符串作为参数。全球化资源不需要被模板所要求。这通常是在应用程序启动时在您的main.js文件（或任何称为应用程序启动文件）中完成的。

用来配置Aurelia的文件将有一个configure方法。Aurelia会调用这个方法，并将Aurelia类的一个实例传递给它。这个类通过它的use属性为访问FrameworkConfiguration对象提供了一个流畅的接口。FrameworkConfiguration对象上的方法返回FrameworkConfiguration对象，使您可以将多个调用链接在一起。

例如：

export function configure(aurelia) {

aurelia.use

.standardConfiguration()

.developmentLogging()

.globalResources('custom-element`, 'another-directory/custom-attribute');

aurelia.start().then(() => aurelia.setRoot());

}

# 在HTML行为中创建可绑定属性

Aurelia的关键特性之一就是强大的数据绑定系统。

@bindable装饰可以从aurelia-framework模块中导入。当把@bindable装饰器用视图模型的一个属性上时，Aurelia将允许使用Aurelia提供的所有绑定命令（bind，two-way，one-way，one-time，等）来绑定到该属性。

例如：

import {bindable} from 'aurelia-framework';

export class GreetCustomerCustomElement {

@bindable customerName = '';

}

<template>

Welcome to Pizza Planet, ${customerName}!

</template>

<template>

<require from="./greet-customer"></require>

<greet-customer customer-name.bind="name"></greet-customer>

</template>

请注意，前面提到的将JavaScript名称转换为破折号的惯例适用于可绑定属性名称。当他们在HTML作为属性出现时，他们会被从驼峰形式转换为dash-case形式。

默认情况下，可绑定属性只允许one-way（单向）数据绑定。这意味着数据会流进HTML行为，而不是从自定义元素流出。幸运的是，这可以被覆盖，或者通过将一个配置对象传递给@bindable装饰器，使用一个名为defaultBindingModeset 的属性设置为bindingMode枚举值之一。此枚举位于在aurelia-framework模块，其有三个值：oneWay，twoWay，和oneTime。

例如：

import {bindable, bindingMode} from 'aurelia-framework';

export class GreetCustomerCustomElement {

@bindable({ defaultBindingMode: bindingMode.twoWay }) customerName = '';

}

自定义属性的绑定相比自定义元素稍有一点差别，因为自定义属性支持三种类型的绑定：单值，选项绑定和动态选项绑定。

在使用自定义属性进行单一值绑定时，@bindable装饰器不会被绑定，因为所有属性默认都有一个value属性。这是由Aurelia确保的。相反，我们实现了一个由Aurelia调用的valueChanged的回调函数，以提醒我们自定义属性的绑定值已更改。Aurelia会将该值设置为自定义属性的视图模型的value属性，并将两个参数传递给valueChanged回调：新值和旧值。

例如：

import {inject} from 'aurelia-framework';

@inject(Element)

export class SquareCustomAttribute {

constructor(element){

this.element = element;

this.element.style.width = this.element.style.height = '100px';

}

valueChanged(newValue, oldValue){

this.element.style.backgroundColor = newValue;

}

}

<template>

<require from="./square"></require>

<div square.bind="color"></div>

</template>

# 三、HTML行为的继承、

对于想要利用继承的开发人员，可绑定属性只能通过自定义元素（不是自定义属性）的类层次结构继承。

例如：

import {bindable} from 'aurelia-templating';

export class IconButton{

@bindable icon = 'ban';

onClick(){

alert("Default method");

}

}

<template>

<button click.delegate="onClick">

<i class="fa fa-${icon}"></i>

</button>

</template>

下一个组件继承了通用按钮，设置其默认图标以及不同的onClick行为

import {useView, customElement} from 'aurelia-templating';

import {IconButton} from './icon-button';

@useView('./icon-button.html')

@customElement(‘add-button’)

export class AddButton extends IconButton {

constructor(){

super();

this.icon = 'plus';

this.onClick = this.add;

}

add(){

alert('Base add button');

}

}

任何时候你继承一个自定义元素，必须添加`customElement`装饰器。

<template>

<require from="./icon-button"></require>

<require from="./add-button"></require>

<icon-button></icon-button>

<icon-button icon="cogs"></icon-button>

<add-button></add-button>

<add-button icon="plus-square-o"></add-button>

</template>

# 只有HTML的自定义元素

早些时候，我们说所有的HTML行为都必须有一个JavaScript类来充当ViewModel（视图模型）规则的一个例外是纯HTML自定义元素。

创建一个只有HTML的自定义元素就像创建一个HTML视图文件一样简单，然后用.html扩展名将它请求到您的视图中，如上面“在视图中创建HTML行为”部分所述。甚至可以在仅HTML自定义元素上具有可绑定属性。这些属性默认为one-way数据绑定，但不能更改默认值，尽管在绑定到自定义元素时仍然可以自由明确的设置绑定方向。要创建可绑定的属性，只需提供一个逗号分隔的属性名称列表到template元素的bindable属性上。

例如：

<template bindable="firstName, lastName">

Hello, ${firstName} ${lastName}!

</template>

<template>

<require from="./hello-world.html"></require>

<hello-world first-name.bind="firstName" last-name.two-way="lastName"></hello-world>

</template>

# 五、HTML行为生命周期

所有HTML行为都有一个定义好的生命周期。使用这个生命周期，您可以点击并触发代码以在适当的时候运行。

constructor()： - 视图模型的构造函数被首先调用。

created(owningView: View, myView: View)：- 如果视图模型实现了created回调，那么接下来会调用它。如果你的行为是一个自定义的元素，它的视图也被创建，并且视图模型和视图都连接到它们的控制器。被创建的回调将接收“owningView”的实例。这是组件在内部声明的视图。如果组件本身有一个视图，将被第二个参数传递。

bind(bindingContext: Object, overrideContext:Object) - 数据绑定是在视图和视图模型上被激活。如果视图模型有一个bind回调，这个时候就会被调用。绑定组件的“绑定内容”作为第一个参数被传递，“重写内容”是被传递的第二个参数。重写内容包含用于遍历父层次结构的信息，也可用于添加组件要添加的任何上下文属性。值得注意的是，当视图模型完成了bind回调时，数据绑定框架将不会为视图模型的可绑定属性调用已更改的处理程序，直到更新这些属性的“下一个”时间。如果您需要对可绑定属性执行特定的后处理，那么在执行bind 回调时，你应该在回调自身内手动执行。

attached() - 接下来，该组件被附加到DOM（在文档中）。如果视图模型有一个attached回调，这个时候就会被调用。

detached() - 在将来的某个时候，组件可能会从DOM中删除。如果/当发生这种情况，并且如果视图模型有一个detached回调，则它将被调用。

unbind() - 组件分离后，通常会解除绑定。如果你的视图模型有unbind回调，它将在这个过程中被调用。

使用生命周期事件就像在行为视图模型类中实现上面的任何方法一样简单。下面是自定义属性使用附加(attached)和分离(detached)回调方法，和封装jQuery插件时常见事项的一个示例：

import {inject} from 'aurelia-framework';

@inject(Element)

export class SomePlugginCustomAttribute {

constructor(element){

this.element = element;

}

attached() {

this.plugin = jQuery(this.element).somePlugin();

}

detached() {

this.plugin.destroy();

}

}