aurelia基础知识--binding

# HTML and SVG Attributes

Aurelia支持绑定HTML和SVG属性到JS表达式中，属性绑定声明有三部分：

attribute.command="expression"

attribute: HTML或SVG属性名

command: aurelia绑定命令值有如下几种:

one-time: 数据流向为从视图模型（view-model）到视图（view），绑定一次。

to-view/one-way: 数据流向为从视图模型（view-model）到视图（view）

from-view: 数据流向为从视图（view）到视图模型（view-model）

two-way: 数据流向为双向

bind: 自动选择绑定模式，通常表单里为双向绑定，其他情况下通常是one-way绑定。

expression:一个js表达式

## 二、DOM Events

绑定系统支持绑定DOM事件。一个DOM事件绑定将在DOM事件被触发时执行一段JS表达式。事件绑定声明有三个部分：

event.command="expression":

event: DOM事件名称，不带“on”前缀

command:Aurelia事件绑定命令:

trigger: 将事件处理直接连接到元素上。当事件被触发时，处理函数将被执行。

delegate: 将一个单独的事件处理程序附加到Document上，当事件处于冒泡时期时，由Document回调合适的关联表达式。

capture: 将一个单独的事件处理程序附加到Doucument上，当事件处于捕捉时期时，由Document回调合适的关联表达式。

expression: 一个js表达式。使用$event属性去连接绑定表达式中的DOM事件。

Aurelia将在delegate和trigger事件处理时会自动调用preventDefault()方法。大多数时，这个行为是你想做的。在你的事件处理函数中返回true可以关闭。

# Function References

当开发自定义元素或自定义属性时，你可能会碰到一种情况就是你使用了@bindable属性希望可以引用一个函数。此时，可以使用“call”绑定命令声明并传递一个函数给绑定属性。Call命令优先于use-case的bind命令，因为它能够在正确的上下文中执行，从而确保this是你所期望的上下文。

你的自定义元素或者自定义属性能够调用被传递到使用标准调用语法绑定到@bindable装饰器绑定的属性属性上的函数this.go()。

如果你需要含参调用某个方法，需要创建一个对象，键是参数名，值是参数值，然后使用参数对象进行调用。参数对象的属性在call绑定表达式中将可以被数据绑定。

# Referencing Elements

使用“ref”绑定命令创建一个引用到DOM元素。ref命令最基本的语法是ref=”expression”。当视图被数据绑定时，指定的表达式将被分配到DOM元素。

ref命令有几个限定符，你可以在自定义元素和自定义属性中结合使用。

* element.ref="expression": 创建一个引用到DOM元素上。 (等同ref="expression").
* attribute-name.ref="expression": 创建一个引用到自定义属性的view-model上。
* view-model.ref="expression": 创建一个引用到自定义元素上。
* view.ref="expression":创建一个引用到自定义元素的视图实例上（并非一个HTML元素）
* controller.ref="expression": 创建一个引用到自定义元素控制器实例上。

# String Interpolation

字符串插值表达式支持用文本表示表达式的结果。字符串差值可以应用到html的属性中，作为对视图绑定的替代。默认情况下，插值绑定的模式是to-view，表达式的结果总是被强制字符串。null或未定义的结果将导致空字符串。

之前说到了字符串插值绑定。插值更易读，但是如果你需要双向绑定一个可编辑的元素。还是需要bind。

# 六、Contextual Properties

绑定系统使您的模板中具有绑定的几个属性，这取决于上下文。

* $this- 绑定上下文(视图模型)。
* $parent- 显式地从组合或重复模板中访问外部作用域. 当当前作用域上的属性掩盖了外部作用域的属性时，可能需要这样做. 例如支持$parent.$parent.foo写法。
* $event- 由trigger或delegate绑定的DOM事件.
* $index- 在一个重复模板中，集合的内容的索引.
* $first- 在重复模板中，如果item是数组中的第一个item，则为true.
* $last- 在重复模板中，如果item是数组中的最后一个item，则为true..
* $even- 在重复模板中，如果这个item的索引编号是偶数，则为true
* $odd- 在重复模板中，如果这个item的索引编号是奇数，则为true

# Class and Style

可以通过使用字符串插值或者.bind/.one-time来绑定一个元素的class。为了保证和其他js框架的兼容性，绑定系统只添加或删除绑定表达式中的class。这确保了其他代码（例如 classList.add(...)）添加的class得以保存。这种“默认安全”行为代价很小，但在基准测试或其他性能关键的情况下，比如重复使用大量元素时，需要注意。

你可以通过使用class-name.bind=””或者class-name.one-time=””来直接绑定元素的className属性，从而退出默认行为。这回稍微快一些，但是可以添加很多额外绑定。

您可以将css字符串或对象绑定到元素的样式属性。在进行字符串插值时，使用样式属性的别名css来 确保您的应用程序与Internet Explorer和Edge兼容



# Checkboxes

Aurelia支持two-way绑定多种数据类型到选择框元素。

1. Booleans

将一个布尔类型绑定到输入框元素的checked属性，使用checked.bind="myBooleanProperty"

1. Array of Numbers

一组复选框元素是一个多选择接口。如果您有一个作为“选择项”列表的数组，您可以将该数组绑定到每个输入的检查属性。绑定系统将会跟踪输入框的checked状态，当输入框被checked时，将输入框的值添加进数组，取消时，移除出数组。想要定义输入框的value，需要绑定输入框的model属性：model.bind="product.id"

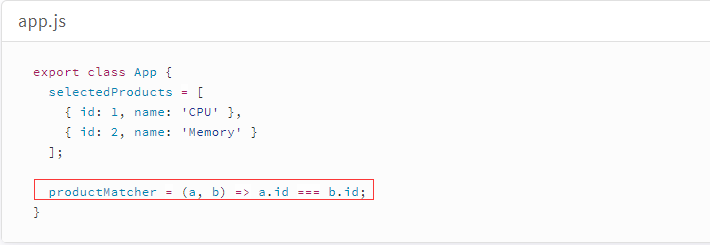
1. Array of Objects

数组并不是唯一可以存储在选择项中数组中的值，绑定系统支持所有类型。包括object。

1. Array of Objects with Matcher

您可能会遇到这样的情况:您的输入元素的模型被绑定到的对象没有对您的检查数组中

的任何对象的引用相等。对象可能与id匹配，但它们可能不是同一个对象实例。为了支持这种情况，你可以覆盖Aurelia的默认“matcher”，这是一个类似于这样的平等比较函数。例如：(a, b) => a === b。你可以用你选择的函数来代替你的对象来比较你的对象。





1. Array of Strings

最后，这里有一个示例，它使用复选框数据绑定从selected生成物数组中添加和删除字符串。这个示例是惟一的，因为它不使用模型。绑定来分配每个复选框的值。而是使用了输入的标准值属性。通常情况下，我们不能将标准值属性与检查绑定结合使用，因为它会强制将其分配给字符串。这个例子使用了一系列字符串。



# Radios

# Selects

# Delegate vs Trigger

最常被问到的问题是delegate和trigger有什么不同，什么时候使用delegate，什么时候使用trigger。最简单的回答是，尽量使用delegate，除非不能使用delegate的时候。事件委托是一种用于提高应用程序性能的技术。它通过利用大多数DOM事件的“冒泡”特性，极大地减少了事件订阅的数量。对于事件委托，处理程序不属于单独的元素。相反，单个事件处理程序被附加到一个顶级节点，比如body元素。当事件冒泡到这个共享顶级处理程序时，事件委托逻辑会根据事件的目标调用适当的处理程序。只有Aurelia的委托绑定命令才能使用的事件。 blur、 focus、 load、unload事件并不冒泡，此时你需要使用trigger

# Computed Properties

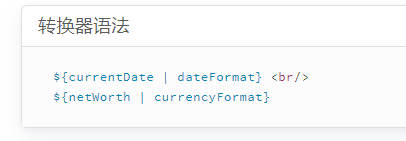
# Value Converters

Value Converter是一个类，职责是将视图模型值转换为适合在视图中显示的值，反之亦然。

Aurelia ValueConverter接口使用toView和fromView方法，这使得它非常清楚数据流向哪个方向。在Aurelia中，转换器参数可以是数据绑定的。Aurelia值转换器方法可以接受多个参数。可以使用管道（|）组合多个值转换器。

Aurelia的一个简单约定：以ValueConverter结尾的导出名称被假定为值转换器。 该协议使用导出名称来注册转换器，驼峰式，从最后剥离ValueConverter部分。

DateFormatValueConverter 注册为 dateFormat



资源在视图中引用的名称源自其导出名称。对于“值转换器”和“绑定行为”，导出名称将转换为骆驼大小写（将其视为变量名称）。对于“自定义元素”和“自定义属性”，导出名称是小写字母和连字符（符合HTML元素和属性规范）

# Binding Behaviors

绑定行为是视图资源的一个类别，就像值转换器，自定义属性和自定义元素一样。绑定行为最想值转换器，在绑定表达式中声明它来影响绑定这一点最像值转换器。

绑定行为和值转换器之间的主要区别是绑定行为在整个生命周期中都可以完全访问绑定实例。和值转换器只能够在值从model传递到view时拦截相比，反之亦然。

绑定行为和值转换器之间的主要区别是绑定行为在整个生命周期中都可以完全访问绑定实例。

1. Throttle

Aurelia会有少量的行为来实现常见的场景。第一个是节流绑定行为，它限制了视图模型在two-way绑定中更新的速度，或者视图在to-view绑定场景中更新的速率。默认情况下，节流阀只允许每200毫秒更新一次。你可以自定义速率。

需要注意的第一件事是“&”符号，它用于声明绑定行为表达式。绑定行为表达式使用与值转换器表达式相同的语法模式。

* 绑定行为可以接收参数:firstName & myBehavior:arg1:arg2:arg3
* 绑定表达式可以包含多个绑定行为:firstName & behavior1 & behavior2:arg1.
* 绑定表达式还可以包括值转换器和绑定行为的组合:${foo | upperCase | truncate:3 & throttle & anotherBehavior:arg1:arg2}



1. debounce

debounce是另一种限制绑定速率的绑定行为。debounce在指定的时间间隔内值没有任何变化前都会阻止绑定的更新。一个常见的用例是搜索输入，它会自动触发搜索。我们通常会希望在用户暂停输入时调用搜索逻辑。Debounce默认200ms。



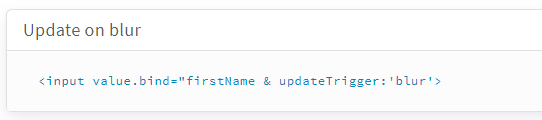
注：throttle与debounce的区别：

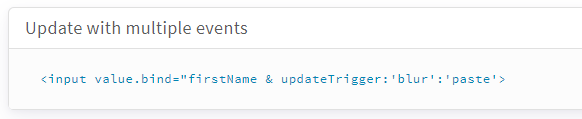
throttles是设置绑定更新的间隔时间。

debounce是设置某段间隔时间，在间隔时间内，值没有变化，就阻止绑定行为。

1. updateTrigger

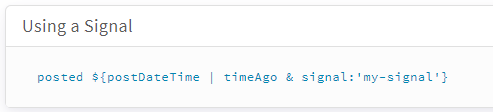
更新触发器允许您覆盖导致将元素的值写入到视图模型的输入事件。默认事件是change和input。例如，如何使绑定只在blur事件中更新model的值。并且updateTrigger支持多事件。





1. Signal

Signal行为使你能够告知绑定去更新。这当绑定结果受到在观察路径之外的全局改变时特别有用。例如，你有一个名为“translate”的值转换器，能够将一个key转换成本地化的字符串（${'greeting-key' | translate}），并且你的网站允许用户修改当前的语言。那么你该如何刷新绑定的值。



在上面的绑定表达式中，我们使用Signal来指定绑定的“signal name”。signal name是任意的，如果你想同时发出多个绑定，你可以给多个绑定一个相同的signal name。下图为如何设置signal绑定周期。

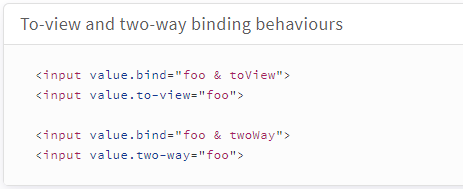


1. oneTime

对于oneTime绑定行为，你可以指定插值绑定行为应该发生一次。绑定模式的名称不同取决于他是绑定命令还是绑定行为。因为HTML不区分大小写，所以绑定命令不能够使用驼峰法，因此绑定命令是用dash-case写法命名。当使用绑定表达式作为绑定行为，由于dash-case写法不是JS变量名命名所允许的，因而使用驼峰法。



这是个很重要的特性。one-Time绑定是最有效的绑定类型，因为它们不会引起任何属性观察开销。这里也有toView和twoWay绑定的用法：

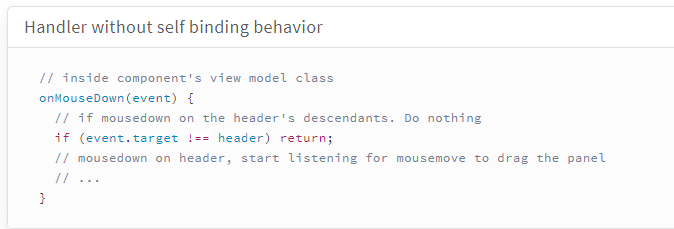


1. Self

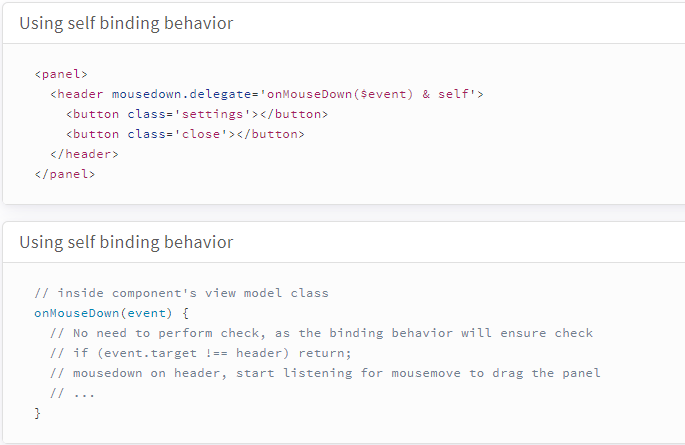
使用self绑定行为，你可以指定事件处理器只监听器关联的目标，而不是他的子节点。例如



onMouseDown是你的事件处理器，他会被不仅鼠标按下header元素，还包括所有他内部的元素所调用。在这个例子中就是settings和close按钮。我们并不总是期望这样。有时你希望组件仅仅响应header而不包括他的按钮。为了实现这些，onMouseDown需要做如下处理：



这样处理虽然有用，但是这不是业务或者组件的逻辑，还是混杂了不必要的DOM事件处理。Self绑定行为可以帮助你实现以上的需求，而不必再你的处理方法里添加不必要的代码。



1. Custom binding behaviors

你可以自定义绑定行为就像自定义值转换器一样。你需要创建bind(binding, scope, [...args])和unbind(binding, scope)方法。在bind方法里，你可以添加你的行为到绑定中，在unbind方法里，你需要清空你在绑定方法里所做的一切从而使得绑定实例恢复到原始状况。“binding”参数是你想改变的行为中需要的绑定实例。它是绑定接口的一个实现。“scope”参数是绑定的数据上下文。他提供了一个访问binding将要被绑定到的模型到他的bindingContext和bindingContext配置。

下面是一个自定义绑定行为每当你的视图模型中的绑定updateSource/updateTarget/callSource方法被调用时，都会调用。

const interceptMethods = ['updateTarget', 'updateSource', 'callSource'];

export class InterceptBindingBehavior {

bind(binding, scope, interceptor) {

let i = interceptMethods.length;

while (i--) {

let method = interceptMethods[i];

if (!binding[method]) {

continue;

}

binding[`intercepted-${method}`] = binding[method];

let update = binding[method].bind(binding);

binding[method] = interceptor.bind(binding, method, update);

}

}

unbind(binding, scope) {

let i = interceptMethods.length;

while (i--) {

let method = interceptMethods[i];

if (!binding[method]) {

continue;

}

binding[method] = binding[`intercepted-${method}`];

binding[`intercepted-${method}`] = null;

}

}

}

<template>

<require from="./intercept-binding-behavior"></require>

<div mousemove.delegate="mouseMove($event) & intercept:myFunc"></div>

<input value.bind="foo & intercept:myFunc">

</template>

# Observable Properties

你是否有当配置改变时，执行一个动作的需求，如果有，配置观察是个很有用的东西。

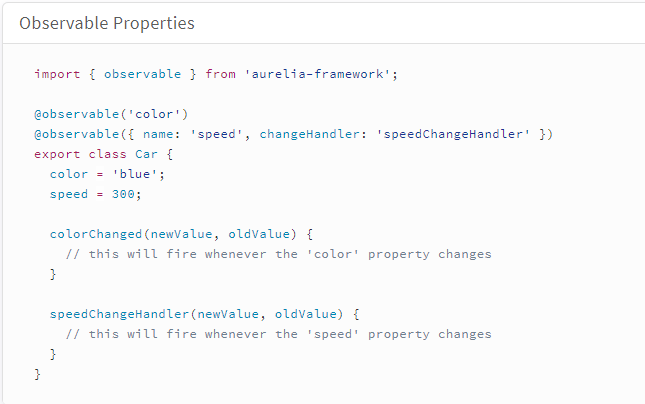
为了观察一个配置，你需要用@observable装饰器声明它，并定义一个方法作为值变化的处理器。这个方法接收2个参数：newValue和oldValue。你可以把业务逻辑写在方法内。

根据约定，值变化处理器的是一个名称由属性名和Changed结尾组成的方法。例如，你要观察color，就需要定义一个colorChanged()方法去处理值改变处理器。



你无须去校验新旧值是否不同，如果被赋予了一个同样的值，值变化处理器不会被调用。如果你不想用约定的命名。你可以通过设 置changeHandler属性来定义回调函数的名称。



如果你喜欢，你也可以把@observable写在class上  


@observable只跟踪属性值的变化，而不是值本身。这意味着，如果属性是数组，当数组添加，删除，编辑items时，值改变处理器不会被调用。