**理解AngularJS的作用域Scope**

**概叙：**

AngularJS中，子作用域一般都会通过JavaScript原型继承机制继承其父作用域的属性和方法。但有一个例外：在directive中使用scope: { ... }，这种方式创建的作用域是一个独立的"Isolate"作用域，它也有父作用域，但父作用域不在其原型链上，不会对父作用域进行原型继承。这种方式定义作用域通常用于构造可复用的directive组件。

作用域的原型继承是非常简单普遍的，甚至你不必关心它的运作。直到你在子作用域中向父作用域的原始类型属性使用双向数据绑定**2-way data binding**，比如Form表单的ng-model为 父作用域中的属性，且为原始类型，输入数据后，它不会如你期望的那样运行——AngularJS不会把输入数据写到你期望的父作用域属性中去，而是直接在 子作用域创建同名属性并写入数据。这个行为符合JavaScript原型继承机制的行为。AngularJS新手通常没有认识到ng-repeat、 ng-switch、ng-view和ng-include 都会创建子作用域, 所以经常出问题。 (见 [示例](http://plnkr.co/edit/zZfUQN?p=preview))

避免这个问题的最佳实践是在ng-model中总使用.，参见文章 [always have a '.' in your ng-models](http://www.youtube.com/watch?v=ZhfUv0spHCY&feature=youtu.be&t=30m)。

比如：

<input type="text" ng-model="someObj.prop1">

优于：

<input type="text" ng-model="prop1">

如果你一定要直接使用原始类型，要注意两点：

1. 在子作用域中使用 $parent.parentScopeProperty，这样可以直接修改父作用域的属性。
2. 在父作用域中定义函数，子作用域通过原型继承调用函数把值传递给父作用域（这种方式极少使用）。

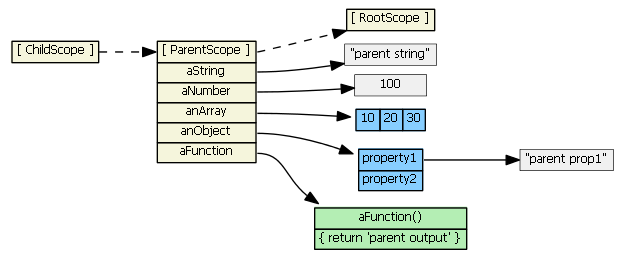
**正文：**

* [JavaScript Prototypal Inheritance](http://www.angularjs.cn/#/A09C#JSproto)
* [Angular Scope Inheritance](http://www.angularjs.cn/#/A09C#Angular)
  + [ng-include](http://www.angularjs.cn/#/A09C#ngInclude)
  + [ng-switch](http://www.angularjs.cn/#/A09C#ngSwitch)
  + [ng-view](http://www.angularjs.cn/#/A09C#ngView)
  + [ng-repeat](http://www.angularjs.cn/#/A09C#ngRepeat)
  + [ng-controller](http://www.angularjs.cn/#/A09C#ngController)
  + [directives](http://www.angularjs.cn/#/A09C#directives)

**JavaScript 原型继承机制**

你必须完全理解JavaScript的原型继承机制，尤其是当你有后端开发背景和类继承经验的时候。所以我们先来回顾一下原型继承：

假设父作用域parentScope拥有以下属性和方法：aString、aNumber、anArray、anObject、aFunction。子作用域childScope如果从父作用域parentScope进行原型继承，我们将看到：



(**注：**为节约空间，anArray使用了蓝色方块图)

如果我们在子作用域中访问一个父作用域中定义的属性，JavaScript首先在子作用域中寻找该属性，没找到再从原型链上的父作用域中寻找，如果还没找到会再往上一级原型链的父作用域寻找。在AngularJS中，作用域原型链的顶端是$rootScope，JavaScript寻找到$rootScope为止。所以，以下表达式均为true：

childScope.aString === 'parent string'

childScope.anArray[1] === 20

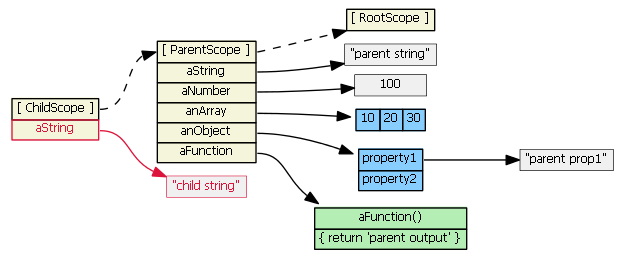
childScope.anObject.property1 === 'parent prop1'

childScope.aFunction() === 'parent output'

如果我们进行如下操作:

childScope.aString = 'child string'

因为我们赋值目标是子作用域的属性，原型链将不会被查询，一个新的与父作用域中属性同名的属性aString将被添加到当前的子作用域childScope中。

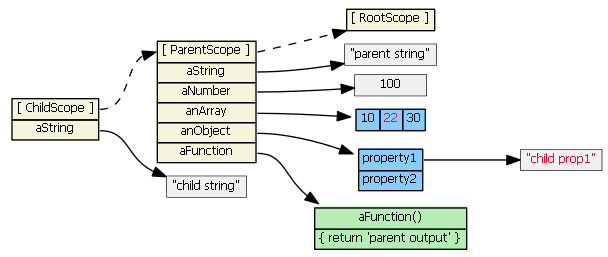


如果我们进行如下操作:

childScope.anArray[1] = '22'

childScope.anObject.property1 = 'child prop1'

因为我们的赋值目标是子作用域属性anArray和anObject的子属性，也就是说JavaScript必须先要先寻找anArray和anObject这两个对象——它们必须为对象，否则不能写入属性，而这两个对象不在当前子作用域，原型链将被查询，在父作用域中找到这两个对象， 然后对这两个对象的属性[1]和property1进行赋值操作。子作用域中不会不会创建两个新的同名属性！（注意JavaScript中数组和函数均是对象——引用类型）

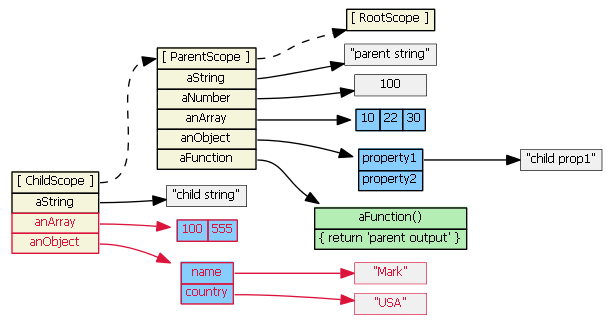


如果我们进行如下操作:

childScope.anArray = [100, 555]

childScope.anObject = { name: 'Mark', country: 'USA' }

同样因为我们赋值目标是子作用域的属性，原型链将不会被查询，，JavaScript会直接在子作用域创建两个同名属性，其值分别为数组和对象。



要点:

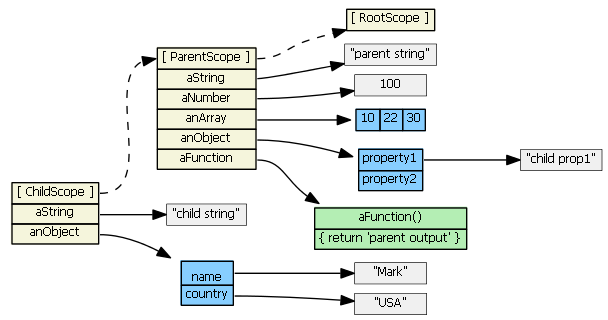
* 如果我们读取childScope.propertyX，并且childScope存在propertyX，原型链不会被查询；
* 如果我们写入childScope.propertyX, 原型链也不会被查询；
* 如果我们写入childScope.propertyX.subPropertyY, 并且childScope不存在propertyX，原型链将被查询——查找propertyX。

最后一点:

delete childScope.anArray

childScope.anArray[1] === 22 // true

如果我们先删除了子作用域childScope的属性，然后再读取该属性，因为找不到该属性，原型链将被查询。



**AngularJS 作用域Scope的继承**

**提示:**

* 以下方式会创建新的子作用域，并且进行原型继承： ng-repeat、ng-include、ng-switch、ng-view、ng-controller, 用scope: true和transclude: true创建directive。
* 以下方式会创建新的独立作用域，不会进行原型继承：用scope: { ... }创建directive。这样创建的作用域被称为"Isolate"作用域。

**注意：**默认情况下创建directive使用了scope: false，不会创建子作用域。

进行原型继承即意味着父作用域在子作用域的原型链上，这是JavaScript的特性。AngularJS的作用域还存在如下内部定义的关系：

* scope.$parent指向scope的父作用域；
* scope.$$childHead指向scope的第一个子作用域；
* scope.$$childTail指向scope的最后一个子作用域；
* scope.$$nextSibling指向scope的下一个相邻作用域；
* scope.$$prevSibling指向scope的上一个相邻作用域；

这些关系用于AngularJS内部历遍，如$broadcast和$emit事件广播，$digest处理等。

**ng-include**

In controller:

$scope.myPrimitive = 50;

$scope.myObject = {aNumber: 11};

In HTML:

<script type="text/ng-template" id="/tpl1.html">

<input ng-model="myPrimitive">

</script>

<div ng-include src="'/tpl1.html'"></div>

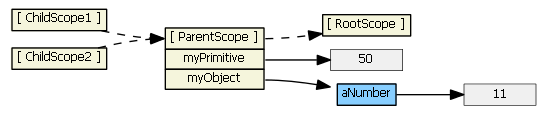
<script type="text/ng-template" id="/tpl2.html">

<input ng-model="myObject.aNumber">

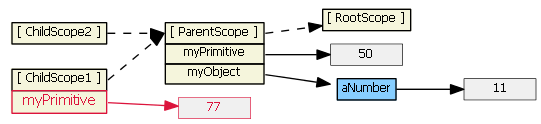
</script>

<div ng-include src="'/tpl2.html'"></div>

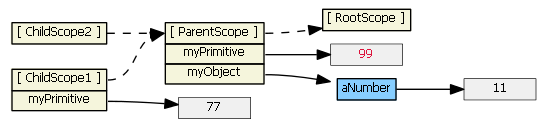
每一个ng-include指令都创建一个子作用域, 并且会从父作用域进行原型继承。



在第一个input框输入"77"将会导致子作用域中新建一个同名属性，其值为77，这不是你想要的结果。



在第二个input框输入"99"会直接修改父作用域的myObject对象，这就是JavaScript原型继承机制的作用。

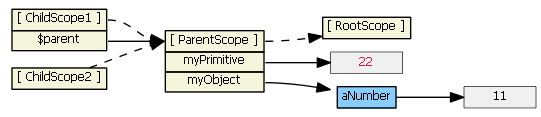


(**注：**上图存在错误，红色99因为是50，11应该是99)

如果我们不想把model由原始类型改成引用类型——对象，我们也可以使用$parent直接操作父作用域：

<input ng-model="$parent.myPrimitive">

输入"22"我们得到了想要的结果。



另一种方法就是使用函数，在父作用域定义函数，子作用域通过原型继承可运行该函数：

// in the parent scope

$scope.setMyPrimitive = function(value) {

$scope.myPrimitive = value;

}

请参考：

[sample fiddle](http://jsfiddle.net/mrajcok/jNxyE/) that uses this "parent function" approach. (This was part of a [Stack Overflow post](http://stackoverflow.com/a/14104318/215945).)

<http://stackoverflow.com/a/13782671/215945>

<https://github.com/angular/angular.js/issues/1267>.

**ng-switch**

ng-switch与ng-include一样。

参考： [AngularJS, bind scope of a switch-case?](http://stackoverflow.com/questions/12405005/angularjs-bind-scope-of-a-switch-case/12414410)

**ng-view**

ng-view与ng-include一样。

**ng-repeat**

Ng-repeat也创建子作用域，但有些不同。

In controller:

$scope.myArrayOfPrimitives = [ 11, 22 ];

$scope.myArrayOfObjects = [{num: 101}, {num: 202}]

In HTML:

<ul><li ng-repeat="num in myArrayOfPrimitives">

<input ng-model="num">

</li>

</ul>

<ul><li ng-repeat="obj in myArrayOfObjects">

<input ng-model="obj.num">

</li>

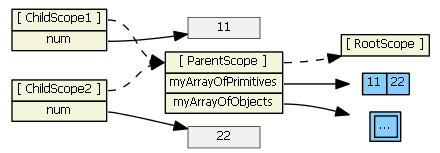
</ul>

ng-repeat对每一个迭代项Item都会创建子作用域, 子作用域也从父作用域进行原型继承。 **但它还是会在子作用域中新建同名属性，把Item赋值给对应的子作用域的同名属性**。 下面是AngularJS中ng-repeat的部分源代码：

childScope = scope.$new(); // child scope prototypically inherits from parent scope ...

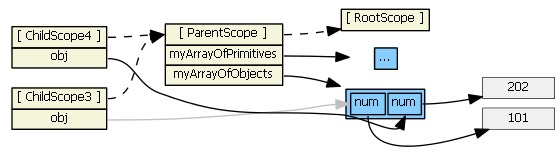
childScope[valueIdent] = value; // creates a new childScope property

如果Item是原始类型(如myArrayOfPrimitives的11、22), 那么子作用域中有一个新属性（如num），它是Item的副本（11、22）. 修改子作用域num的值将**不会**改变父作用域myArrayOfPrimitives，所以在上一个ng-repeat，每一个子作用域都有一个num 属性，该属性与myArrayOfPrimitives无关联：



显然这不会是你想要的结果。我们需要的是在子作用域中修改了值后反映到myArrayOfPrimitives数组。我们需要使用引用类型的Item，如上面第二个ng-repeat所示。

myArrayOfObjects的每一项Item都是一个对象——引用类型，ng-repeat对每一个Item创建子作用域，并在子作用域新建obj属性，obj属性就是该Item的一个引用，而不是副本。



我们修改子作用域的obj.num就是修改了myArrayOfObjects。这才是我们想要的结果。

参考：

[Difficulty with ng-model, ng-repeat, and inputs](http://stackoverflow.com/questions/13714884/difficulty-with-ng-model-ng-repeat-and-inputs)

[ng-repeat and databinding](http://stackoverflow.com/a/13782671/215945)

**ng-controller**

使用ng-controller与ng-include一样也是创建子作用域，会从父级controller创建的作用域进行原型继承。但是，利用原型继承来使父子controller共享数据是一个糟糕的办法。 ["it is considered bad form for two controllers to share information via $scope inheritance"](http://onehungrymind.com/angularjs-sticky-notes-pt-1-architecture/)，controllers之间应该使用 service进行数据共享。

(如果一定要利用原型继承来进行父子controllers之间数据共享，也可以直接使用。 请参考： [Controller load order differs when loading or navigating](http://stackoverflow.com/questions/13825419/controller-load-order-differs-when-loading-or-navigating/13843771#13843771))

**directives**

1. 默认 (scope: false) - directive使用原有作用域，所以也不存在原型继承，这种方式很简单，但也很容易出问题——除非该directive与html不存在数据绑定，否则一般情况建议使用第2条方式。
2. scope: true - directive创建一个子作用域, 并且会从父作用域进行原型继承。 如果同一个DOM element存在多个directives要求创建子作用域，那么只有一个子作用域被创建，directives共用该子作用域。
3. scope: { ... } - directive创建一个独立的“Isolate”作用域，没有原型继承。这是创建可复用directive组件的最佳选择。因为它不会直接访问/修改 父作用域的属性，不会产生意外的副作用。这种directive与父作用域进行数据通信有如下四种方式（更详细的内容请参考[Developer Guide](http://docs.angularjs.org/guide/directive" \t "_blank)）：
   1. **= or =attr** “Isolate”作用域的属性与父作用域的属性进行双向绑定，任何一方的修改均影响到对方，这是最常用的方式；
   2. **@ or @attr** “Isolate”作用域的属性与父作用域的属性进行单向绑定，即“Isolate”作用域只能读取父作用域的值，并且**该值永远的String类型**；
   3. **& or &attr** “Isolate”作用域把父作用域的属性包装成一个函数，从而以函数的方式读写父作用域的属性，包装方法是$parse，详情请见[API-$parse](http://docs.angularjs.org/api/ng.$parse" \t "_blank)；

“Isolate”作用域的\_\_proto\_\_是一个标准[Scope](http://docs.angularjs.org/api/ng.$rootScope.Scope" \t "_blank) object (the picture below needs to be updated to show an orange 'Scope' object instead of an 'Object'). “Isolate”作用域的$parent同样指向父作用域。它虽然没有原型继承，但它仍然是一个子作用域。

如下directive：

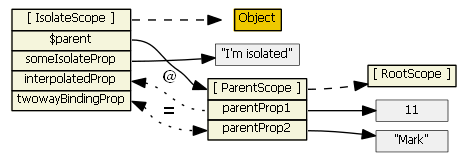
<my-directive interpolated="{{parentProp1}}" twowayBinding="parentProp2">

scope：

scope: { interpolatedProp: '@interpolated', twowayBindingProp: '=twowayBinding' }

link函数中：

scope.someIsolateProp = "I'm isolated"

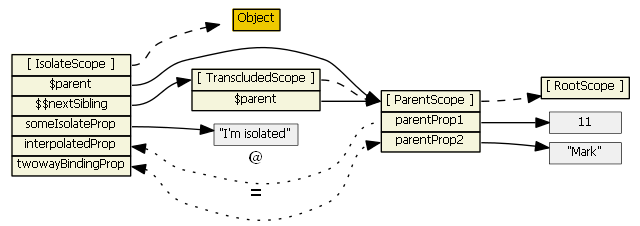


请注意，我们在link函数中使用attrs.$observe('interpolated', function(value) { ... }来监测@属性的变化。

更多请参考： <http://onehungrymind.com/angularjs-sticky-notes-pt-2-isolated-scope/>

1. transclude: true - directive新建一个“transcluded”子作用域，并且会从父作用域进行原型继承。需要注意的是，“transcluded”作用域与 “Isolate”作用域是相邻的关系（如果“Isolate”作用域存在的话） -- 他们的$parent属性指向同一个父作用域。“Isolate”作用域的$$nextSibling指向“transcluded”作用域。

更多请参考： [AngularJS two way binding not working in directive with transcluded scope](http://stackoverflow.com/a/14484903/215945)



demo: [fiddle](http://jsfiddle.net/mrajcok/7g3QM/)

**总结**

AngularJS存在四种作用域:

1. 普通的带原型继承的作用域 -- ng-include, ng-switch, ng-controller, directive with scope: true；
2. 普通的带原型继承的，并且有赋值行为的作用域 -- ng-repeat，**ng-repeat**为每一个迭代项创建一个普通的有原型继承的子作用域，但同时在子作用域中创建新属性存储迭代项；
3. “Isolate”作用域 -- directive with scope: {...}， 该作用域没有原型继承，但可以通过'=', '@', 和 '&'与父作用域通信。
4. “transcluded”作用域 -- directive with transclude: true，它也是普通的带原型继承的作用域，但它与“Isolate”作用域是相邻的好基友。

Diagrams were generated with [GraphViz](http://graphviz.org) "\*.dot" files, which are on [github](https://github.com/mrajcok/angularjs-prototypal-inheritance-diagrams). Tim Caswell's ["Learning JavaScript with Object Graphs"](http://howtonode.org/object-graphs) was the inspiration for using GraphViz for the diagrams.

The above was originally posted on [StackOverflow](http://stackoverflow.com/questions/14049480/what-are-the-nuances-of-scope-prototypal-prototypical-inheritance-in-angularjs).