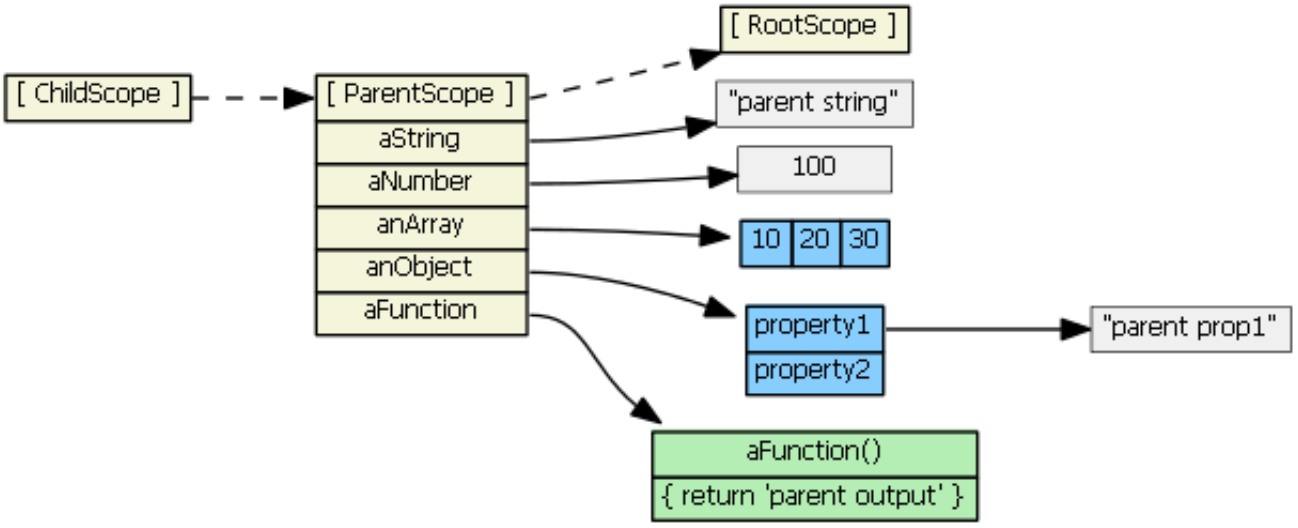
JavaScript Prototypal Inheritance

首先我们先来回顾以下 javascript 中出现的原型继承:

```
function ParentScope(){
    this.aString = "parent string";
    this.aNumber = 100;
    this.anArray = [10,20,30];
    this.anObject = {
    'property1': 'parent prop1',
    'property2': 'parent prop2'
    };
    this.aFunction = function(){
      return 'parent output';
    }
}

function ChildScope(){
}

ChildScope.prototype = new ParentScope();

var childScope = new ChildScope();
```

ChildScope 原型继承自 ParentScope



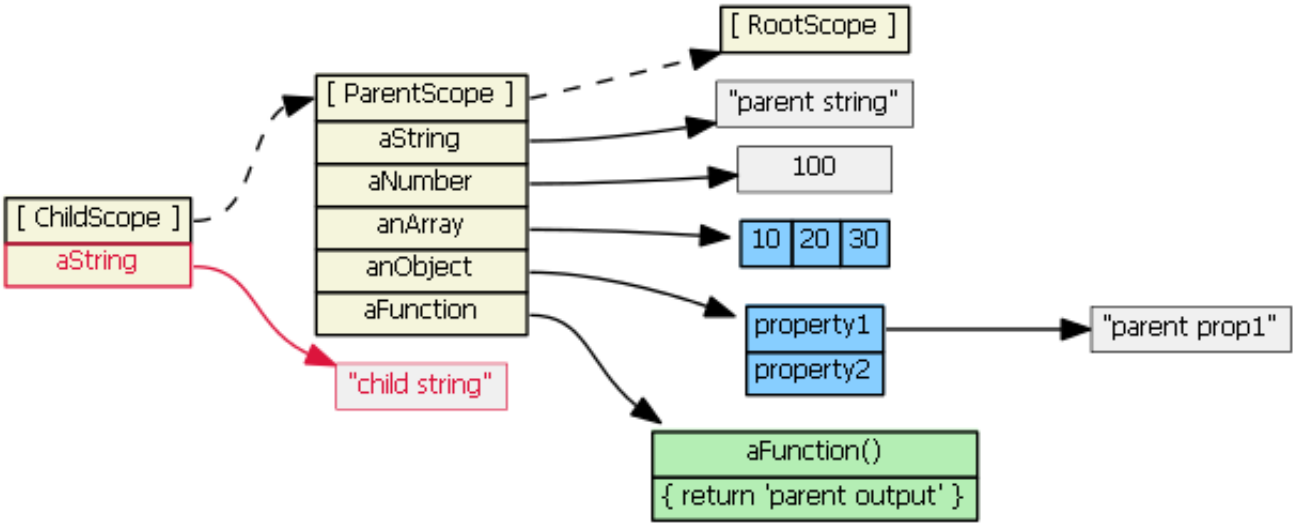
如果我们要在 childScope 上查询一个定义在 parentScope 的属性, JavaScript 会先在 childScope 上查找, 如果没有查到, 那么会顺着原型链去查找. 所以以下判别式均为 `true`

```
childScope.aString === 'parent string'
childScope.anArray[1] === 20
childScope.anObject.property1 === 'parent prop1'
childScope.aFunction() === 'parent output'
```

如果我们做如下操作:

```
childScope.aString = 'child string'
```

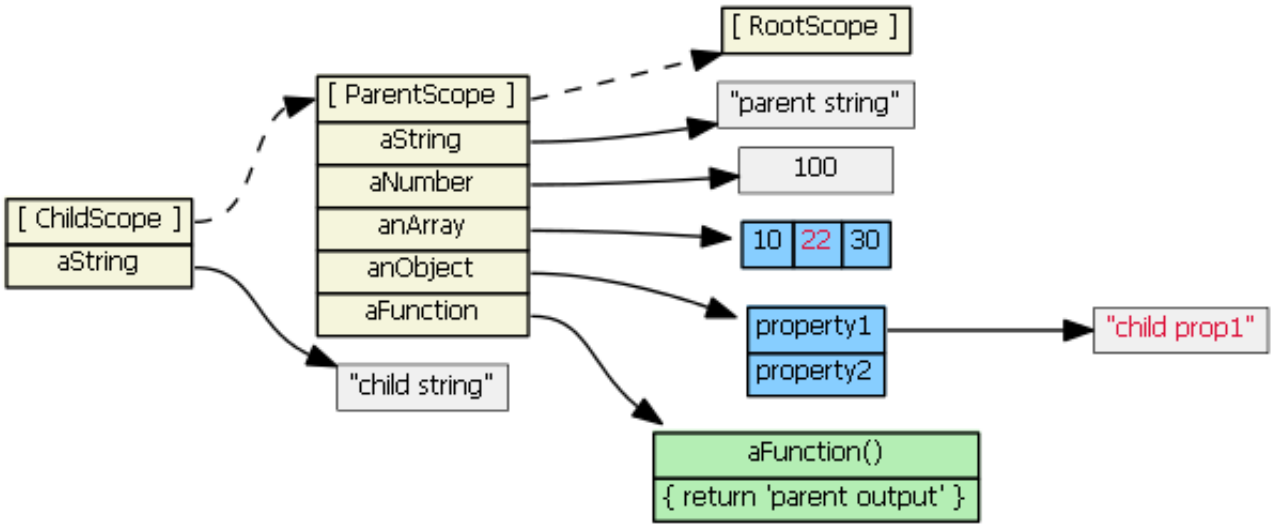
原型链并没有被访问, 一个新的 `aString` 会被加入到 childScope 的属性中去, 新的属性会隐藏 parentScope 中的同名属性.



假设我们做出如下操作:

```
childScope.anArray[1] = 22
childScope.anObject.property1 = 'child prop1'
```

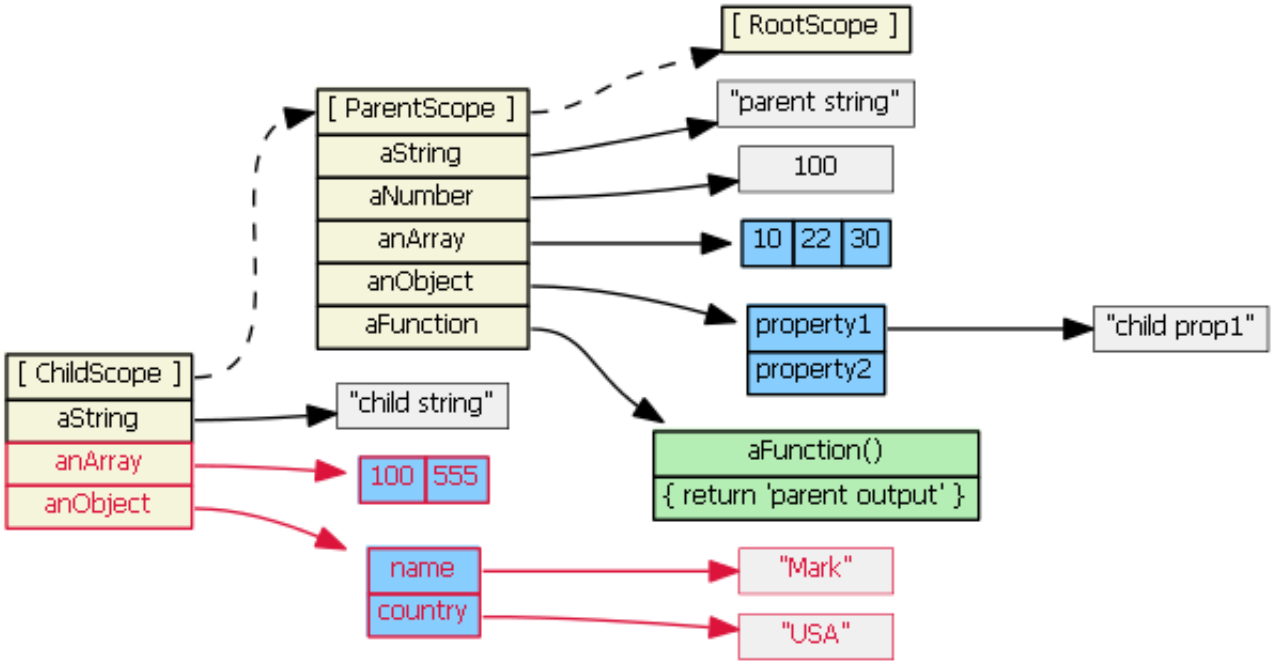
原型链被访问了. 因为 `anArray` , `anObject` 没有在 childScope 中找到.
新的赋值操作均在 parentScope 上进行. childScope 上没有添加任何新的属性.



如果我们做出如下操作

```
childScope.anArray = [100, 555]
childScope.anObject = { name: 'Mark', country: 'USA' }
```

原型链没有被访问, childScope 会获得两个新的属性, 并且会隐藏 parentScope 上的同名属性.



仔细体会上面的三次操作. 第一第三次均是对某个属性进行赋值, 原型链并不会被访问, 由于属性并不存在, 所以新的属性将会被添加. 而第二次其实是先访问, `childScope.anArray` , `childScope.anObject` , 再对其访问的对象的某个属性进行复制.

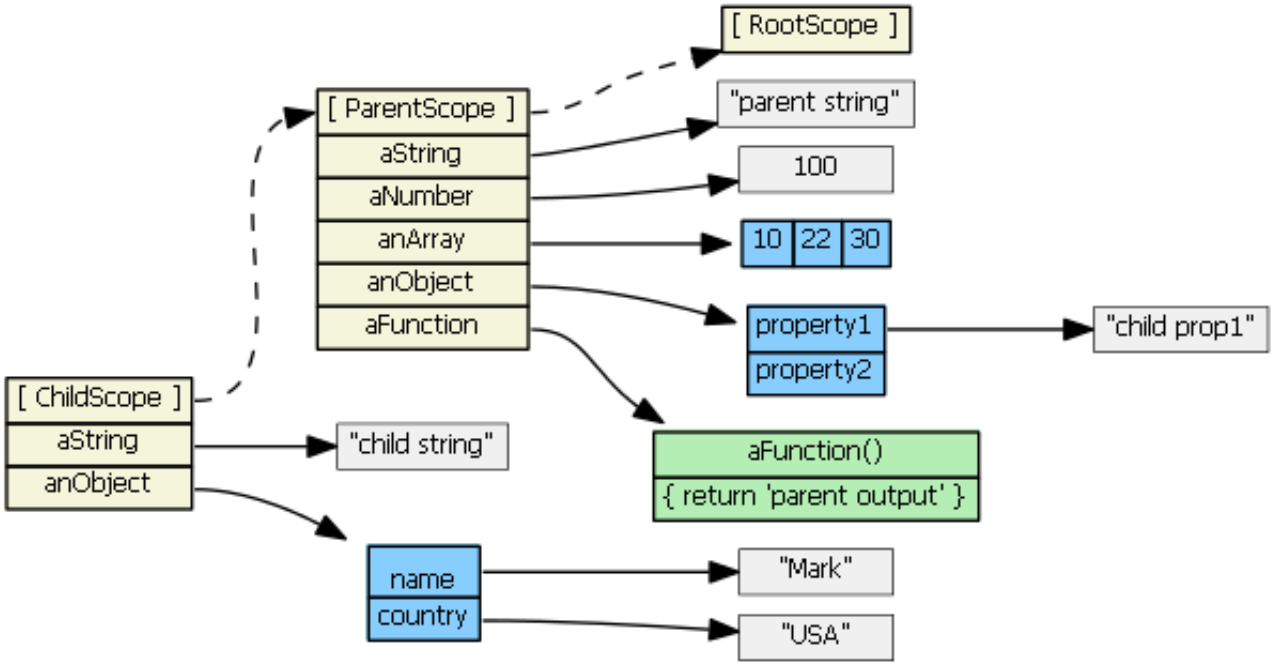
总结:

- 如果我们读取 childScope.propertyX, 而 childScope 拥有 propertyX, 那么原型链不会被访问
- 如果我们读取 childScope.propertyX, 而 childScope 并没有 propertyX, 那么原型链会被访问.
- 如果对 childScope.propertyX 进行赋值, 那么原型链并不会被访问.

最后我们再来看一种情况:

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

我们显示删除了 childScope 的一个属性, 接着试图读取这个属性, 由于 childScope 并没有了这个属性, 所以原型链会被访问.



Angular Scope Inheritance

接着我们来看看 Angular 中的 scope 继承

以下指令会创建新的 scope, 并且会在原型上继承 父scope (即$scope.$parent, 下文两个词互为同义词):

- ng-repeat
- ng-switch
- ng-view
- ng-controller
- 带有 `scope: true` 的指令
- 带有 `transclude: true` 的指令

以下指令创建新的指令, 且在原型上 **不继承** 父scope:

- 带有 `scope: { ... }` 的指令, 这会创建一个 独立的scope (isolate scope)

注意: 默认指令并不会创建 scope, 默认是 `scope: false` , 通常称之为 共享scope.

让我们来看几个例子:

ng-include

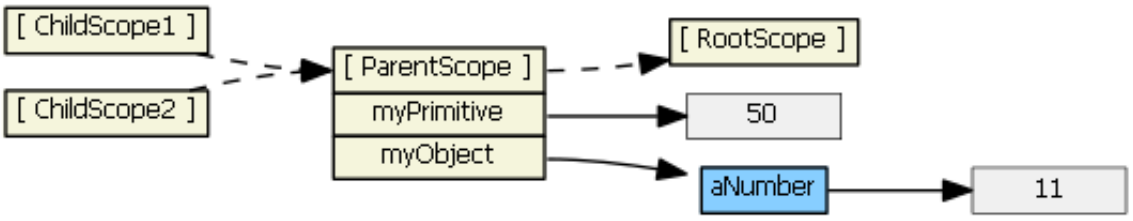
JS:

```
$scope.myPrimitive = 50;
$scope.myObject    = {aNumber: 11};
```

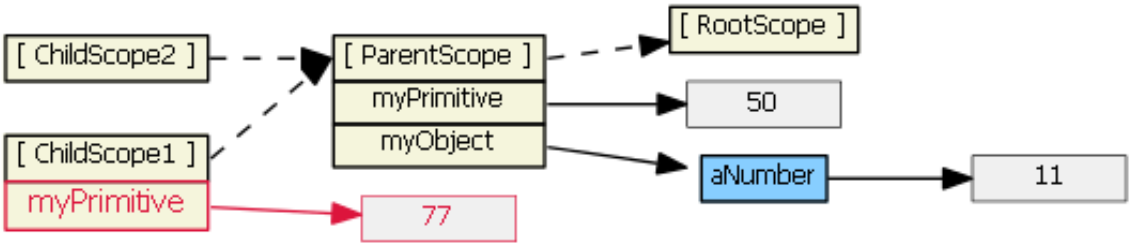
HTML:

```
<p>{ myPrimitive }</p>
<p>{ myObject.aNumber }</p> // cannot use double curly brackets in jekyll

<script type="text/ng-template" id="/tpl1.html">
    <input type="number" ng-model="myPrimitive">
</script>
<div ng-include src="'/tpl1.html'"></div>

<script type="text/ng-template" id="/tpl2.html">
    <input type="number" ng-model="myObject.aNumber">
</script>
<div ng-include src="'/tpl2.html'"></div>
```

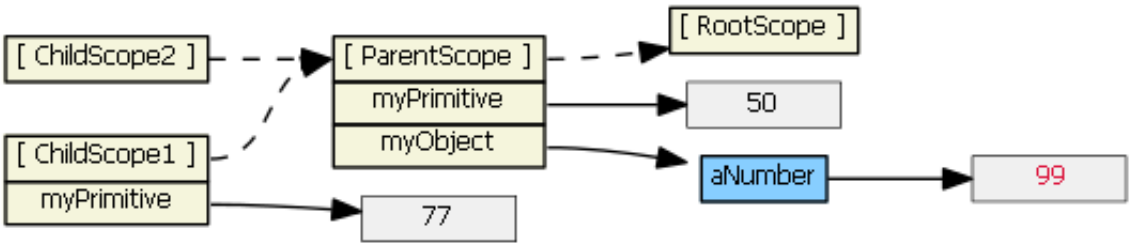
每一个 ng-include 都会创建一个 子scope, 并在原型上继承 父 scope



向第一个 input 输入数字, 一个新的属性 `myPrimitive` 将会被创建, 同时隐藏 父 scope 的 `myPrimitive`;



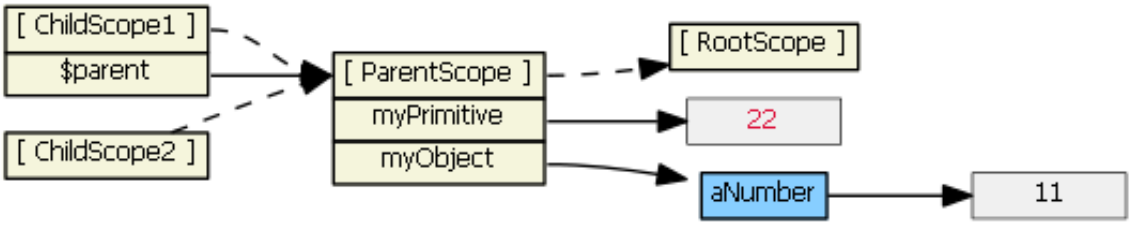
向第二个 input 输入数字, 子 scope 并不会创建一个新的属性, 这时候原型继承发挥了作用.



第一种情况很可能不是我们期待的结果, 所以可以显式的调用 `$parent` 来解决这个问题.

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

向第一个 input 键入数字, 这时候就不会产生新的属性了. $parent 指向了 父scope. 但是 $parent 和 原型上的继承并不一定相等. 稍后我们会看到一个例子.



对于所有的 scope, 无论是共享的(`scope: false`), 继承的(`scope: true`), 还是孤立的 (`scope: { ... }`), Angular 都会建立一个 父-子 的层级关系, 这个层级关系是根据 dom 结构的层级关系决定的, 可以通过 `$parent` , `$$childHead` , `$$childTail` 来访问.

为了避免刚才的例子出现的子 scope 创建新属性情况的发声, 除了使用 `$scope` , 还可以使用调用原型链上的方法.

```
// in the parent scope
$scope.setMyPrimitive = function(value) {
    $scope.myPrimitive = value;
}
```

ng-switch ng-view

ng-switch, ng-view 与 ng-include 情况类似, 不赘述.

ng-repeat

ng-repeat 有一点特殊.

JS:

```
$scope.myArrayOfPrimitives = [ 11, 22 ];
$scope.myArrayOfObjects    = [{num: 101}, {num: 202}]
```

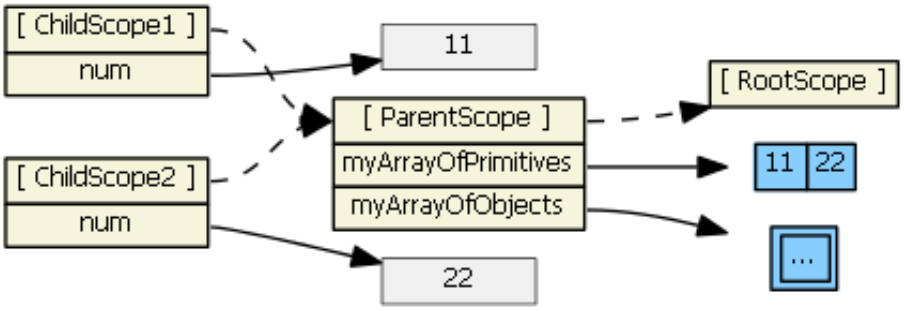
HTML:

```
<ul><li ng-repeat="num in myArrayOfPrimitives">
       <input ng-model="num"></input>
    </li>
</ul>
<ul><li ng-repeat="obj in myArrayOfObjects">
       <input ng-model="obj.num"></input>
    </li>
</ul>
```

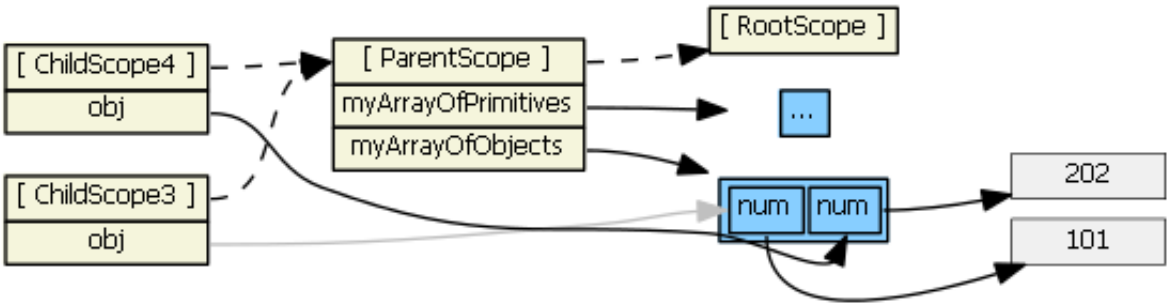
对于每一次迭代, ng-repeat 都会创建一个 子scope, 并在原型上继承 父scope, **但是他还会将 父**scope **上的属性赋值到 子**scope **上**. 新的属性名就是 `ng-repeat="** in parentScope.property"` 中的 **.
源码中的 ng-repeat 是这样的:

```
childScope = scope.$new(); // child scope prototypically inherits from parent scope ...
childScope[valueIdent] = value; // creates a new childScope property
```

如果 ** 是 primitive, 那么一份 copy 会被赋值到新的属性上. 修改 子scope 上的新属性自然不会修改 父 scope 上的属性.



如果 ** 是个 object, 那么一个 reference 会被赋值到新的 子scope 属性上. 修改这个属性, 就是修改 父scope 对应的属性.



ng-controller

ng-controller 也是会创建新的 子scope, 同时原型继承 父scope. 如同 ng-include, ng-switch, ng-view.
但是, 使用 $scope 来共享数据被认为是一种不好的操作. 因为原型链可是会一直向上追溯的.
如果想要共享数据, 最好使用 service.

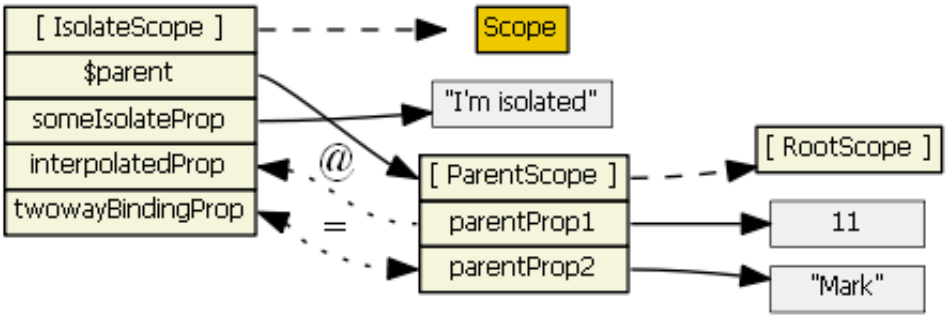
Angular Directives

我们来总结以下指令中的 scope:

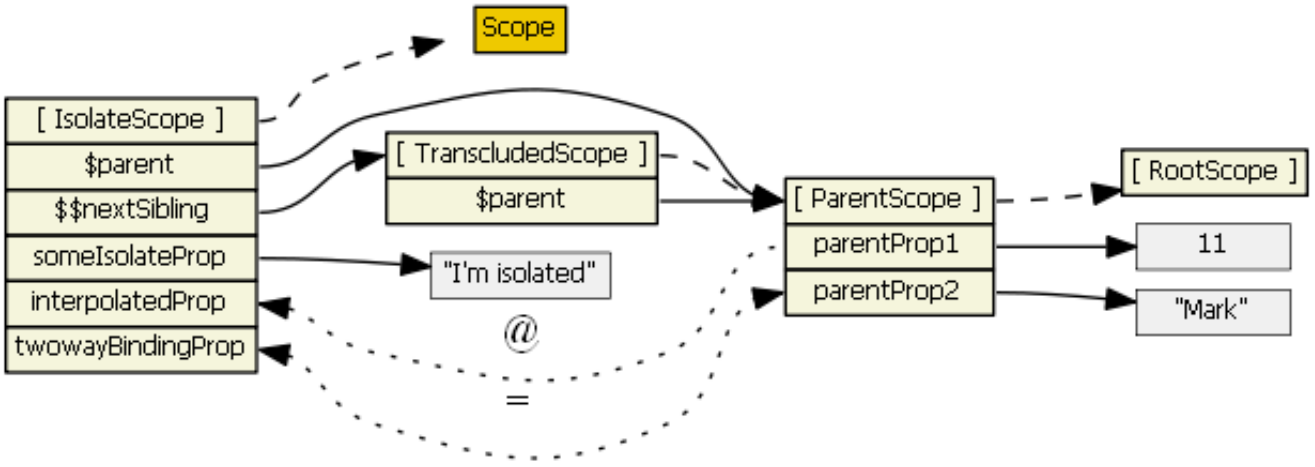
1. `scope: false` (默认的), 指令不会创建新的 scope, 没有继承关系. 与 $parent 共享 $scope.
2. `scope: true` , 指令会创建一个 子scope, 并在原型上继承 $parent. 如果在一个 DOM 上有多个指令想要创建新的 scope, 会报错.
3. `scope: { ... }` , 指令会创建一个 孤立的scope. 这在创建可重用的组件时是最好的选择. 但是, 即使如此, 指令还是希望读取 $parent 的数据. 这个时候可以使用如下符号获得:
    - `scope: { **: "="}` 与 $parent 建立双向绑定.
    - `scope: { **: "@"}` 与 $parent 建立单向绑定.
    - `scope: { **: "&"}` 绑定 $parent 的表达式.
      想要获得相应的属性, 必须通过指令上的属性获得
    - HTML: `<div my-directive the-Parent-Prop=parentProp>`
    - JS: `scope: { localProp: '@theParentProp' }`
      假设:
    - HTML: `<my-directive interpolated="" twowayBinding="parentProp2">`
    - JS: `scope: { interpolatedProp: '@interpolated', twowayBindingProp:`

```
'=twowayBinding' }
```

- 指令在 link 期间: `scope.someIsolateProp = "I'm isolated"`
  其中的关系如图:



4. `transclude: true` , 指令创建了一个 “transcluded” 的子scope, 在原型上继承其 父scope. 如果上述例子同时具有 `transclude: true` . 那么这个 “transcluded” scope, 和 “islolated” scope 是姊妹关系. 他们的 $parent 指向同一个 scope. 且 isolate scope 的 $$nextSibling 就是这个 “transcluded scope”. 下图反应了他们之间的关系:



Reference

- Angular Wiki: Understanding Scopes (https://github.com/angular/angular.js/wiki/Understanding-Scopes)