ANGULARJS实例教程（二）

作用域与事件

徐飞 @民工精髓V

MVVM

- 模型
- 视图
- 视图模型
- 作用域是三者的结合部分

根作用域

- 每个Angular应用都有默认的作用域
- 这个作用域是整个应用的根
- 可以通过注入$rootScope的方式访问到
- 如果一个应用内未定义控制器，界面模板中定义的变量会赋值在根作用域上

作用域的继承

- 一般每个控制器实例都会注入自己的作用域
- 如果两个控制器实例绑定的界面模板存在包含关系，内层对应的控制器所注入的作用域与外层作用域自动存在继承关系
- 这种继承是通过JavaScript原型实现的

示例

```
<div ng-controller="OuterCtrl">
    <span>{{a}}</span>
    <div ng-controller="InnerCtrl">
        <span>{{a}}</span>
    </div>
</div>
function OuterCtrl($scope) {
    $scope.a = 1;
}
function InnerCtrl($scope) {}
```

简单变量的赋值

- 上一页的代码里补充一点，这个按钮点击之后，上下两级结果分别是什么?

```
<div ng-controller="OuterCtrl">

    <span>{{a}}</span>

    <div ng-controller="InnerCtrl">

        <span>{{a}}</span>

        <button ng-click="a=a+1">a++</button>

    </div>

</div>
```

通过对象引用实现共享

- 如果我们使用引用，是可以实现上下级作用域之间的数据共享的

```
<div ng-controller="OuterCtrl">
    <span>{{data.a}}</span>
    <div ng-controller="InnerCtrl">
        <span>{{data.a}}</span>
        <button ng-click="data.a=data.a+1">increase a</button>
    </div>
</div>
function OuterCtrl($scope) {
    $scope.data = {a: 1};
}
```

控制器实例别名

- 在模板绑定语法中，应当尽量使用限定命名来减少混乱

- Angular1.2版本之后，控制器可以不需要注入作用域，可以使用实例别名来区分变量

示例

```
<div ng-controller="CtrlB as instanceB">
    <div>{{instanceB.a}}</div>
    <button ng-click="instanceB.foo()">click me</button>
</div>
function CtrlB() {
    this.a = 1;
    this.foo = function() {};
}
```

显式指定上级作用域

- 每个作用域上都有$parent，指向它的原型，也就是父级作用域，可以通过它来访问上级的属性和方法

```
<div ng-controller="OuterCtrl">
    <span>{{a}}</span>
    <div ng-controller="InnerCtrl">
        <span>{{a}}</span>
        <button ng-click="$parent.a=a+1">increase a</button>
    </div>
</div>
```

不请自来的新作用域

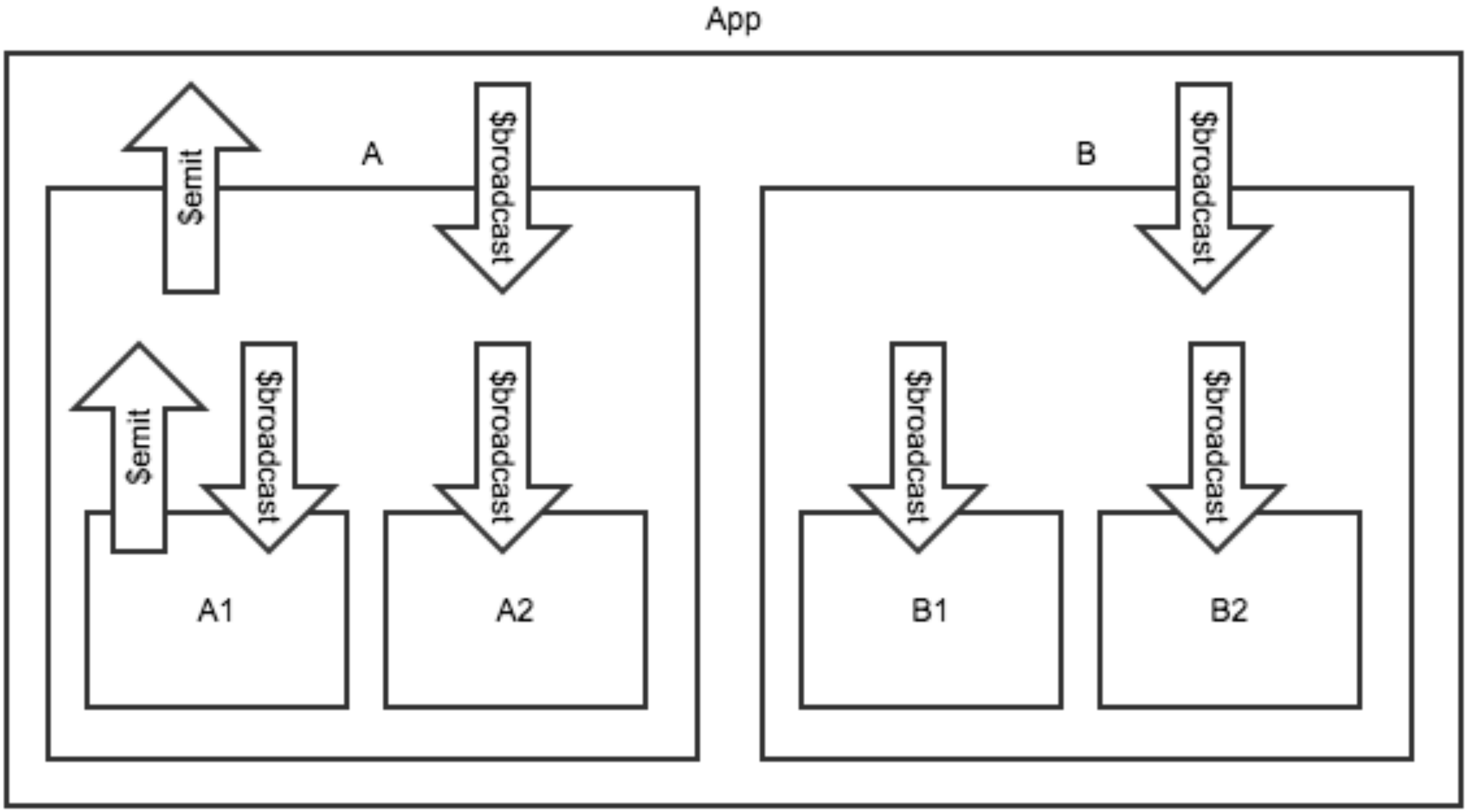
- 并非只有ng-controller会引入新作用域
- ng-repeat，ng-if，ng-switch，ng-include等内置指令都会引入新的作用域
- 注意：$parent是用于访问上一级作用域的，并非用于访问上级控制器的

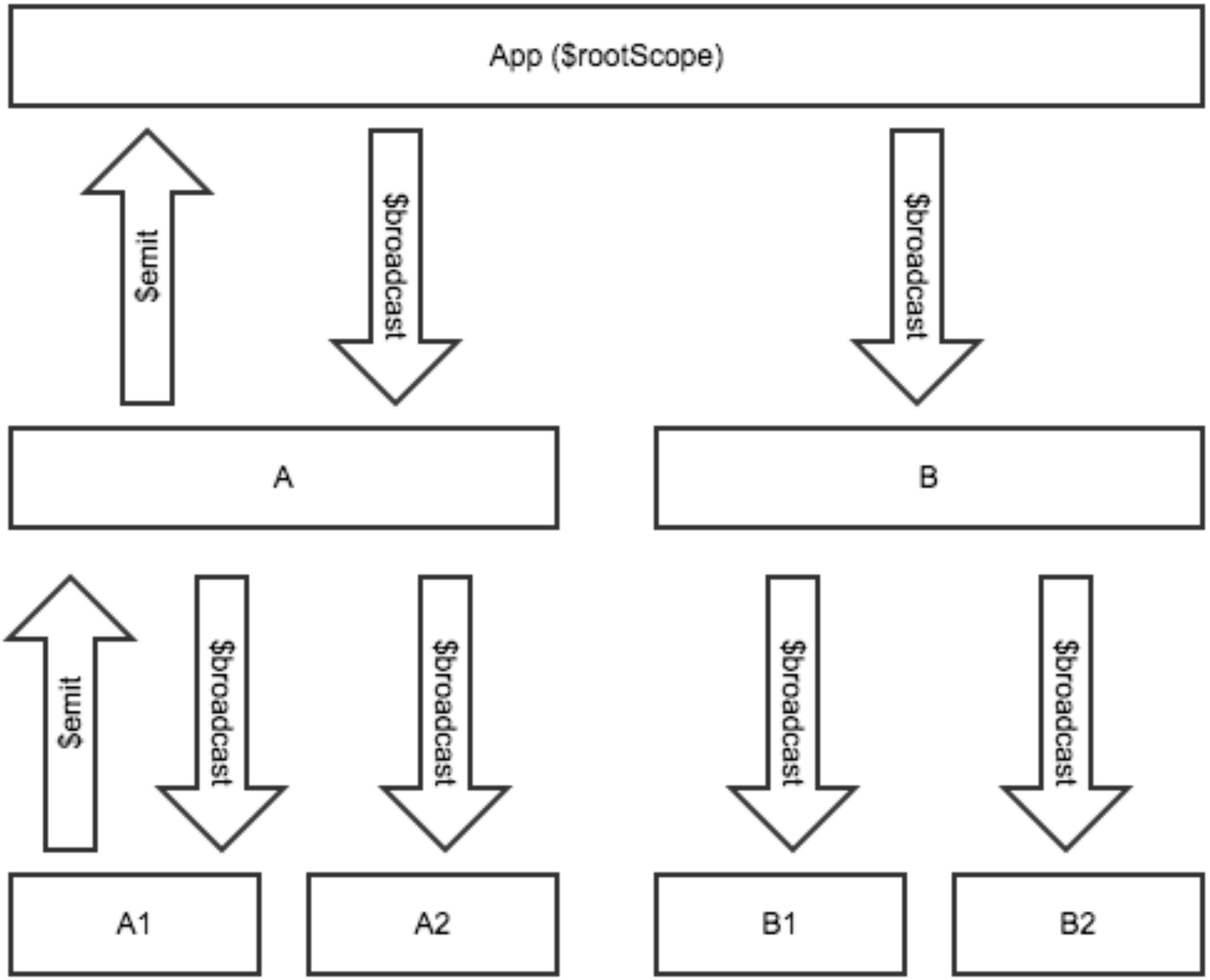
“悬空”作用域

- 并不是所有作用域都一定要关联到界面上
- 可以在已有的作用域上调用$new以创建新作用域，该作用域将会成为新作用域的$parent
- 这个新作用域与界面并无关联，可以通过$compile去跟界面进行关联
- 这种不关联界面的“悬空”作用域可以只用来当作可监控的数据对象使用，仍然可以调用$watch等方法

作用域事件

- 作用域事件是脱离DOM而存在的，用于发送业务通知
- $emit沿着作用域链向根作用域方向发送事件
- $broadcast从所选作用域向其子孙作用域发送事件


App
$emit
A
$broadcast
$emit
$broadcast
$broadcast
A1
A2
B
$broadcast
$broadcast
$broadcast
B1
B2

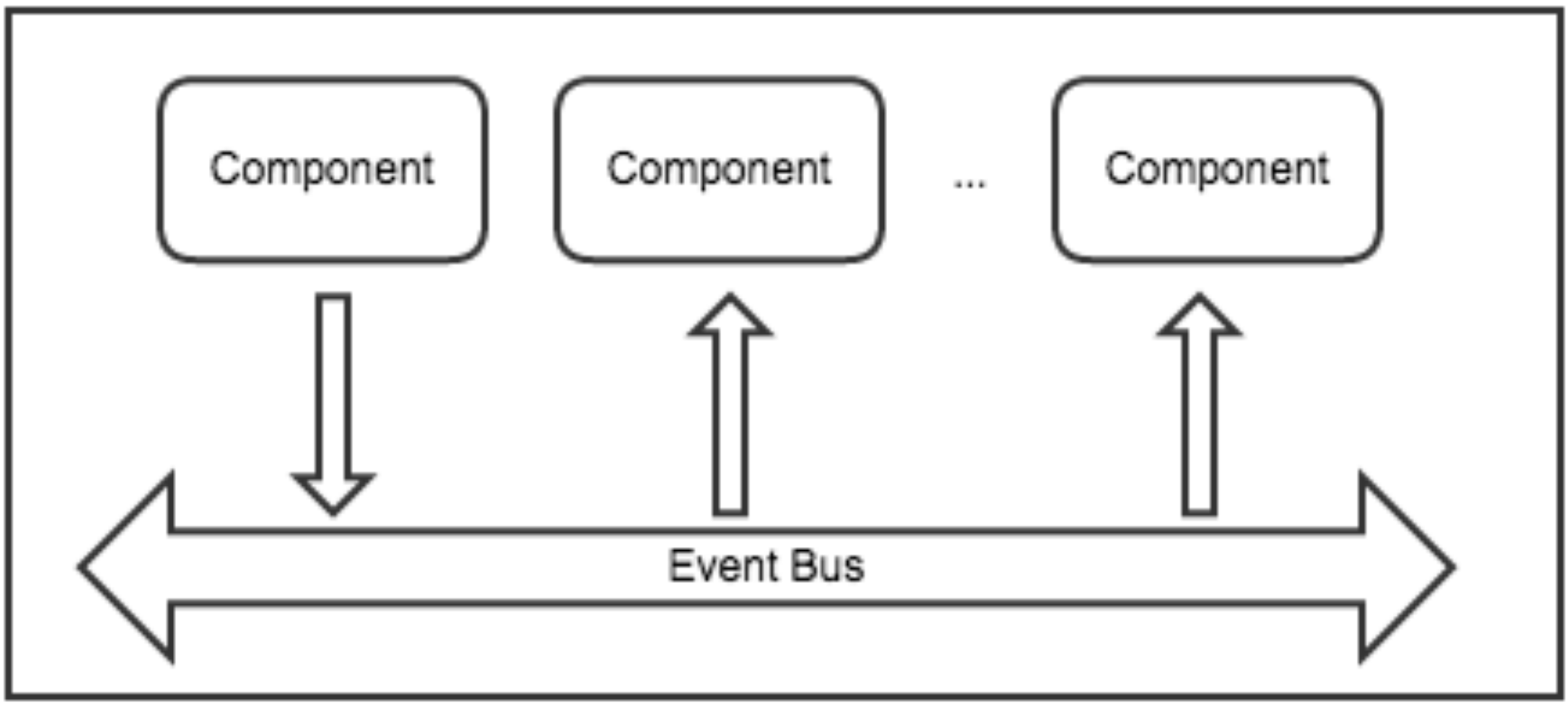

App ($rootScope)
$emit
$broadcast
$broadcast
A
B
$emit
$broadcast
$broadcast
$broadcast
$broadcast
A1
A2
B1
B2

事件的传播与阻止

- 与DOM事件类似，作用域事件也可以被阻止传播
- e.stopPropagation()可以阻止emit事件继续传播
- broadcast事件不能阻止传播，只能阻止默认行为，然后在下一级事件处理函数中判断

事件总线

- 内置事件是一种组件间的通信方式，但是效率偏低，尤其当通信的双方在作用域树上的距离较远的时候
- 可以使用service来进行通信
- 创建一个订阅/发布模式的服务，作为事件总线来进行通信，可以把通信过程扁平化


Component
Component
...
Component
Event Bus

使用事件的意义

- 当应用逐渐大型化，事件的使用愈加重要
- 事件是组件解耦后，组件间必备的通信机制
- 使用事件总线的时候，要注意规划事件名避免冲突

Q && A

首要问题不是自由，而是建立合法的公共秩序。人类可以无自由而有秩序，但不能无秩序而有自由。

——缪尔·亨廷顿

教程地址

https://github.com/xufei/blog/issues/18