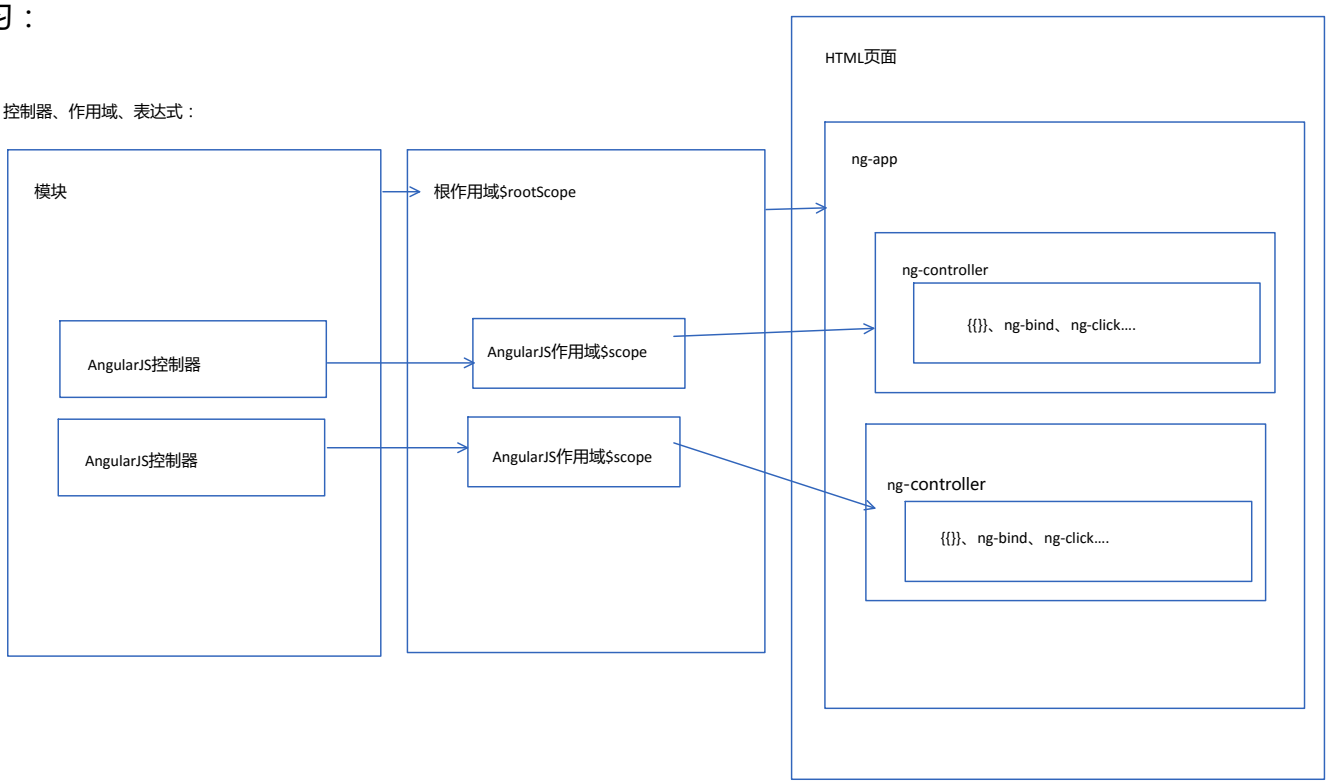
复习：

模块、控制器、作用域、表达式：



---

ng-repeat:

用于根据数组或者对象循环生成HTML标签。

 {{item.name}} , {{item.age}} (ng-repeat指令绑定数组)

 {{key}} , {{value}} (ng-repeat指令绑定对象)

AngularJS服务

2016年5月13日 21:00

在模块上指定**服务名**和**服务对象**（可以是对象、函数、数值等等任意JavaScript变量），在AngularJS框架中其它用得到回调函数的地方，可以**使用依赖注入的方式获得这个服务对象**。用法：

```
module.factory( 'demoService', function(){
    // Todo: 进行一些运算
    return {
        name: 'demo',
        fn: function(){ ... }
    }
})

module.controller('demoController', function( $scope, demoService ){

})
```

创建服务的其他方法：

1.
```
module.service(
    'demoService',
    function(){
        this.name = "demo";
        this.xxx = xxx
        ....
    }
)
```

service和factory唯一的区别就是，service的回调函数不返回一个对象，而是把自身当做一个构造函数使用，创建出来的对象作为服务对象。

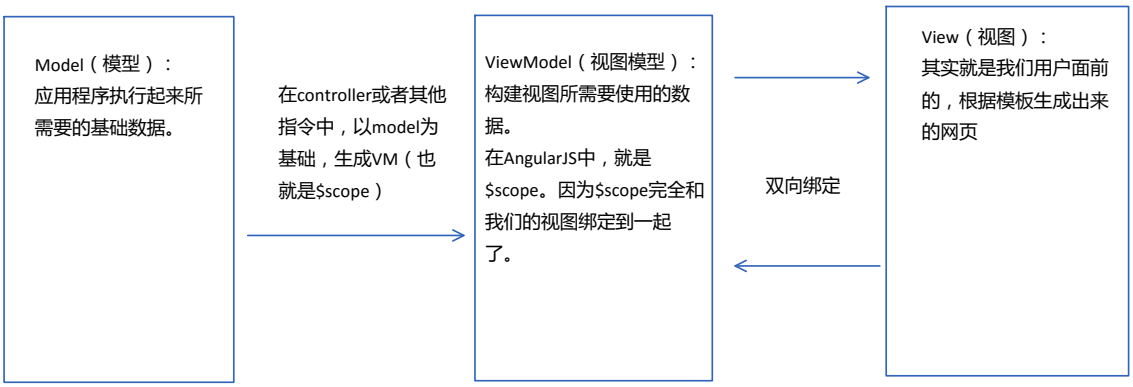
2.
```
module.provider('demoService',function(){
    var name = 'demo'
    return {
        $get : function(){
            return name;
        },
        setName:function(newName){
            name = new Name;
        }
    }
})
```

使用provider时，其回调函数返回的对象中，$get就是我们的服务对象。至于其他字段，我们可以在module.config里访问。如下：

```
module.config(function(demoServiceProvider){
    demoServiceProvider.setName = 'demoPlus'
})
```

名单app的制作

我们先来复习一下MVVM的结构：



实际上AngularJS更像是个MVVM框架，那么为什么还有MVC的说法呢？因为我们可以手动的在VM上把data和action分离开来，让事件处理相关的内容更加独立。当然，会被称作是MVC框架实际上这也和AngularJS的一些历史有关。
总之，AngularJS本身更加适合做MVVM，但是真正做成什么，其实还是看程序员是如何看待它的。无论MVC还是MVVM、它们都是设计思想，只是告诉你程序应该分成哪些部分，不涉及到具体如何去写程序。

如何以更加清晰的方式设计我们的程序？

1. 先来分析业务逻辑：
   a. 显示邀请名单
   b. 新增邀请人
   c. 修改邀请人的受邀请状态
   d. 删除受邀请人
2. 分析过业务内容之后，就可以确定Model：
   a. 在今天之前，为了方便大家理解，我一直在说Model就是数据。实际上，这并不是完全准确的，Model不仅仅是数据，也是数据的访问。真正的Model，我们可能做成一个对象，它内部有数据的同时，还有访问、获取数据的各种业务逻辑。
   b. 确定Model是什么东西之后，我们可以把Model封装成一个服务，供其他部分去使用。
3. 确定了Model之后，我们就来确定界面和ViewModel（因为绑定机制，它们应该是一起的）：
   a. 我们的界面分成两部分：
      i. 上部分的新增邀请
      ii. 下部分的邀请名单
   b. 我们可以分别作出两个控制器，分别绑定两个部分，提供两个vm。然后两个控制器访问同一个model服务，这样不相关的功能就可以尽量分离开。
4. 总结，一个程序拿过来我应该怎么设计：
   a. 第一步：分析业务逻辑，确定Model
   b. 第二步：分析界面，把整个页面分解成一个一个的“HTML标签+VM”，然后一步一步的完成。

一个完善的Model会包含什么？

1. 数据内容
2. 操作数据的各种方法，比如：
   a. 新增邀请人的方法
   b. 删除邀请人的方法
   c. 更改邀请人状态的方法
   d. 返回所有邀请人的方法
   e. 返回所有“邀请中”的邀请人的方法
   f. 返回所有“已接受”的邀请人的方法
   g. 返回所有“已拒绝”的邀请人的方法
   h. ……
3. 总而言之，我们可以把原始数据封装一层，提供一些“返回处理过的数据”的接口给程序的其他部分使用。（实际上所谓的“业务逻辑”不就是对数据的各种处理吗）

AngularJS自定义指令

AngularJS的指令就是自定义的各种标记。

创建指令：

```
directives.directive('demoHello',function(){

    // 返回一个对象，这个对象用于描述我们的指令
    return {
        restrict:"EA", // 设置这个指令可以接受元素、和属性的标记
        template:"<div>this is demo-hello directive</div>"
    }
});
```

★ 如果没有指定restrict属性，默认是“EA”

使用指令：

```
<h3>用标签打标记：</h3>
<demo-hello></demo-hello>

<h3>用属性打标记：</h3>
<div demo-hello></div>
```

当restrict的字符串里面有E的时候，可以用标签名来打标记

当restrict的字符串中有A的时候，可以用属性名来打标记

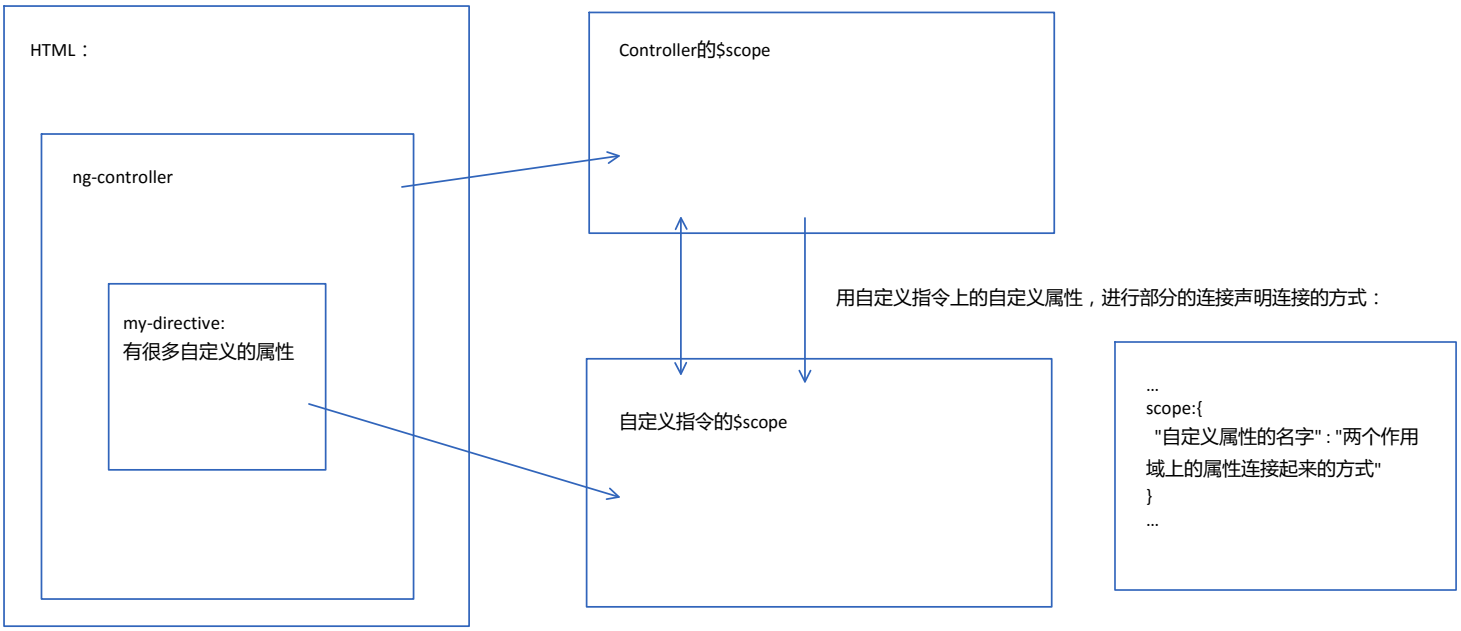
---

templateUrl: 模板的路径（对应的是template传入模板的字符串）

transclude：内嵌功能。原来的标签的内容，放入到模板中指定的标签之内。

replace：替换，用模板中的元素，替换指令所指定的那个元素。

自定义指令的作用域


HTML：
ng-controller
my-directive:
有很多自定义的属性
Controller的$scope
自定义指令的$scope
用自定义指令上的自定义属性，进行部分的连接声明连接的方式：
...
scope:{
"自定义属性的名字" : "两个作用域上的属性连接起来的方式"
}
...