http://www.cnblogs.com/chaojidan/p/4241253.html  
name - 当前scope的名称，注册时可以使用默认值（不填）。

priority（优先级）- 当有多个directive定义在同一个DOM元素时，有时需要明确它们的执行顺序。这属性用于在directive的compile function调用之前进行排序。如果优先级相同，则执行顺序是不确定的（经初步试验，优先级高的先执行，同级时按照类似栈的“后绑定先执行”。另外，测试时有点不小心，在定义directive的时候，两次定义了一个相同名称的directive，但执行结果发现，两个compile或者link function都会执行）。

terminal（最后一组）- 如果设置为”true”，则表示当前的priority将会成为最后一组执行的directive。任何directive与当前的优先级相同的话，他们依然会执行，但顺序是不确定的（虽然顺序不确定，但基本上与priority的顺序一致。当前优先级执行完毕后，更低优先级的将不会再执行）。

scope - 如果设置为：

true - 将为这个directive创建一个新的scope。如果在同一个元素中有多个directive需要新的scope的话，它还是只会创建一个scope。新的作用域规则不适用于根模版（root of the template），因此根模版往往会获得一个新的scope。

{}(object hash) - 将创建一个新的、独立(isolate)的scope。”isolate” scope与一般的scope的区别在于它不是通过原型继承于父scope的。这对于创建可复用的组件是很有帮助的，可以有效防止读取或者修改父级scope的数据。这个独立的scope会创建一个拥有一组来源于父scope的本地scope属性(local scope properties)的object hash。这些local properties对于为模版创建值的别名很有帮助（useful for aliasing values for templates -\_-!）。本地的定义是对其来源的一组本地scope property的hash映射(Locals definition is a hash of local scope property to its source #&)$&@#)($&@#\_)：

@或@attr - 建立一个local scope property到DOM属性的绑定。因为属性值总是String类型，所以这个值总是返回一个字符串。如果没有通过@attr指定属性名称，那么本地名称将与DOM属性的名称一直。例如<widget my-attr=”hello {{name}}”>，widget的scope定义为：{localName:’@myAttr’}。那么，widget scope property的localName会映射出”hello {{name}}"转换后的真实值。name属性值改变后，widget scope的localName属性也会相应地改变（仅仅单向，与下面的”=”不同）。name属性是在父scope读取的（不是组件scope）

=或=expression（这里也许是attr） - 在本地scope属性与parent scope属性之间设置双向的绑定。如果没有指定attr名称，那么本地名称将与属性名称一致。例如<widget my-attr=”parentModel”>，widget定义的scope为：{localModel:’=myAttr’}，那么widget scope property “localName”将会映射父scope的“parentModel”。如果parentModel发生任何改变，localModel也会发生改变，反之亦然。（双向绑定）

&或&attr - 提供一个在父scope上下文中执行一个表达式的途径。如果没有指定attr的名称，那么local name将与属性名称一致。例如<widget my-attr=”count = count + value”>，widget的scope定义为：{localFn:’increment()’}，那么isolate scope property “localFn”会指向一个包裹着increment()表达式的function。一般来说，我们希望通过一个表达式，将数据从isolate scope传到parent scope中。这可以通过传送一个本地变量键值的映射到表达式的wrapper函数中来完成。例如，如果表达式是increment(amount)，那么我们可以通过localFn({amount:22})的方式调用localFn以指定amount的值（上面的例子真的没看懂，&跑哪去了？）。

controller - controller 构造函数。controller会在pre-linking步骤之前进行初始化，并允许其他directive通过指定名称的require进行共享（看下面的require属性）。这将允许directive之间相互沟通，增强相互之间的行为。controller默认注入了以下本地对象：

$scope - 与当前元素结合的scope

$element - 当前的元素

$attrs - 当前元素的属性对象

$transclude - 一个预先绑定到当前转置scope的转置linking function :function(cloneLinkingFn)。(A transclude linking function pre-bound to the correct transclusion scope)

require - 请求另外的controller，传入当前directive的linking function中。require需要传入一个directive controller的名称。如果找不到这个名称对应的controller，那么将会抛出一个error。名称可以加入以下前缀：

? - 不要抛出异常。这使这个依赖变为一个可选项。

^ - 允许查找父元素的controller

restrict - EACM的子集的字符串，它限制directive为指定的声明方式。如果省略的话，directive将仅仅允许通过属性声明：

E - 元素名称： <my-directive></my-directive>

A - 属性名： <div my-directive=”exp”></div>

C - class名： <div class=”my-directive:exp;”></div>

M - 注释 ： <!-- directive: my-directive exp -->

template - 如果replace 为true，则将模版内容替换当前的HTML元素，并将原来元素的属性、class一并迁移；如果为false，则将模版元素当作当前元素的子元素处理。想了解更多的话，请查看“Creating Widgets”章节（在哪啊。。。Creating Components就有。。。）

templateUrl - 与template基本一致，但模版通过指定的url进行加载。因为模版加载是异步的，所以compilation、linking都会暂停，等待加载完毕后再执行。

replace - 如果设置为true，那么模版将会替换当前元素，而不是作为子元素添加到当前元素中。（注：为true时，模版必须有一个根节点）

transclude - 编译元素的内容，使它能够被directive所用。需要(在模版中)配合ngTransclude使用(引用)。transclusion的优点是linking function能够得到一个预先与当前scope绑定的transclusion function。一般地，建立一个widget，创建isolate scope，transclusion不是子级的，而是isolate scope的兄弟。这将使得widget拥有私有的状态，transclusion会被绑定到父级（pre-isolate）scope中。（上面那段话没看懂。但实际实验中，如果通过<any my-directive>{{name}}</any my-directive>调用myDirective，而transclude设置为true或者字符串且template中包含<sometag ng-transclude>的时候，将会将{{name}}的编译结果插入到sometag的内容中。如果any的内容没有被标签包裹，那么结果sometag中将会多了一个span。如果本来有其他东西包裹的话，将维持原状。但如果transclude设置为’element’的话，any的整体内容会出现在sometag中，且被p包裹）

true - 转换这个directive的内容。（这个感觉上，是直接将内容编译后搬入指定地方）

‘element’ - 转换整个元素，包括其他优先级较低的directive。（像将整体内容编译后，当作一个整体（外面再包裹p），插入到指定地方）

compile - 这里是compile function，将在下面实例详细讲解

link - 这里是link function ，将在下面实例详细讲解。这个属性仅仅是在compile属性没有定义的情况下使用。