AngularJS 概念概述(Conceptual Overview)

在创建第一个应用程序之前，你需要理解Angular中的一些概念。 在本章，通过一个简单的例子，你可以快速接触Angular中所有重要的部分。 但是，我不会花时间去解释细节，而是着重于帮助你建立一个全局性的“概观”。 更深入的解释，请参见AngularJS-PhoneCat (../phonecat)。

概念	**说明**
模板(Template) (../tutorial/13.html)	带有Angular扩展标记的**HTML**
指令(Directive) (../tutorial/5.html)	用于通过**自定义属性和元素**扩展HTML的行为
模型(Model)	用于显示给用户并且与用户互动的**数据**
作用域(Scope) (../tutorial/12.html)	用来存储模型(Model)的语境(context)。模型放在这个语境中才能被控制器、指令和表达式等访问到
表达式(Expression) (../tutorial/3.html)	模板中可以通过它来访问作用域（Scope）中的变量和函数
编译器(Compiler) (../tutorial/15.html)	用来**编译**模板(Template)，并且对其中包含的指令(Directive)和表达式(Expression)进行**实例化**
过滤器(Filter) (../tutorial/8.html)	负责**格式化**表达式(Expression)的值，以便呈现给用户

概念	**说明**
视图(View)	用户看到的内容（即**DOM**）
数据绑定(Data Binding) (../tutorial/10.html)	**自动同步**模型(Model)中的数据和视图(View)表现
控制器(Controller) (../tutorial/2.html)	视图(View)背后的**业务逻辑**
依赖注入(Dependency Injection) (../tutorial/17.html)	负责创建和自动装载对象或函数
注入器(Injector)	用来实现依赖注入(Injection)的容器
模块(Module) (../tutorial/6.html)	用来配置注入器
服务(Service) (../tutorial/19.html)	独立于视图(View)的、**可复用**的业务逻辑

例子的第一步：数据绑定

下面我们将编写一个表单，用来计算一个订单在不同币种下的总价。

首先，我们做一个单币种表单，它有数量和单价两个输入框，并且把数量和单价相乘得出该订单的总价。

源码

index.html ()

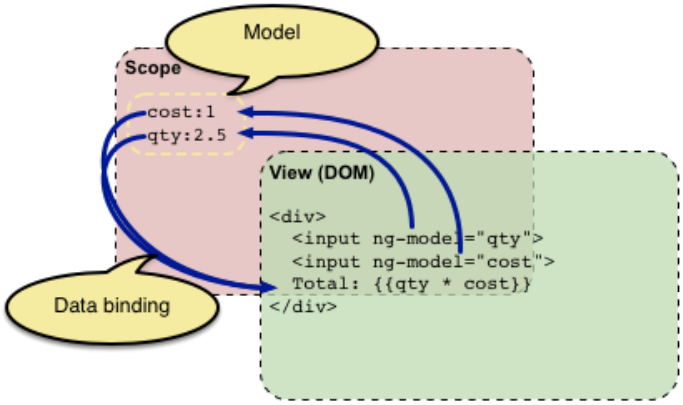
```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
  </head>
  <body>
    <div ng-init="qty=1;cost=2">
      <b>订单:</b>
      <div>
        数量: <input type="number" ng-model="qty" required >
      </div>
      <div>
        单价: <input type="number" ng-model="cost" required >
      </div>
      <div>
        <b>总价:</b>
      </div>
    </div>
  </body>
</html>
```

效果

Invoice:
Quantity: 1
Costs: 2
Total: $2.00

你可以先在Demo区体验一下操作，接下来我们将通读这个例子，并且详细讲解这里所发生的事情。

这看起来很像标准的HTML，只是带了一些新的标记。在Angular中，像这样的文件叫做“模板(template) (../tutorial/13.html)”。 当Angular启动你的应用时，它通过“编译器(compiler) (../tutorial/15.html)”来解析并处理模板中的这些新标记。 这些经过加载、转换、渲染而成的DOM就叫做“视图(view)”



第一类新标记叫做“指令(directive) (../tutorial/17.htmlrective)”。 它们通过HTML中的属性或元素来为页面添加特定的行为。上面的例子中，我们使用了 `ng-app` (api/ng.directive:ngApp) 属性，与此相关的指令(directive)则负责自动初始化我们的应用程序。 Angular还为 `input` (api/ng.directive:input) 元素定义了一个指令，它负责添加额外的行为到这个元素上。例如，当它发现了 `input` 元素的 `required` 属性，

它就会自动进行验证，确保所输入的文本不会是空白。 `ng-model` (api/ng.directive:ngModel)指令则负责从变量(比如这里的qty、cost等)加载 `input` 元素的 `value` 值，并且把 `input` 元素的 `value` 值写回变量中。并且，还会根据对 `input` 元素进行校验的结果自动添加相应的css类。 比如在上面这个例子中，我们就通过这些css类把空白 `input` 元素的边框设置为红色，来表示无效的输入。

通过自定义指令访问DOM: 对于Angular，一个程序中唯一允许接触DOM的地方就是“指令”。之所以这样要求，是因为需要访问DOM的代码难以进行自动化测试。 如果你需要直接访问DOM，那么你就应该为此写一个自定义指令。在 指令指南(../tutorial/17.htmlrective)一章中详细解释了该怎样实现自定义指令。

第二类新标记是双大括号 `{{ expression | filter }}` ，其中expression是“表达式”语句，filter是“过滤器”语句。 当编译器遇到这种标记时，它会把这些标记替换为这个表达式的计算结果。 模板中的"表达式 (../tutorial/3.html)"是一种**类似于**JavaScript的代码片段，它允许你读写变量。注意，表达式中所用的变量**并不是**全局变量。 就像JavaScript函数定义中的变量都属于某个作用域一样，Angular也为这些能从表达式中访问的变量提供了一个“作用域(scope) (../tutorial/12.html)”。 这些存储于Angular作用域(Scope)中的变量叫做Scope变量，这些变量所代表的数据叫做“模型(model)”。 在上面的例子中，这些标记告诉Angular：“从这两个 `input` 元素中获取数据，并把它们乘在一起。”

上面这个例子中还包含一个"过滤器(filter) (../tutorial/8.html)"。 过滤器格式化表达式的值，以便呈现给用户。 上面的例子中 `currency` (api/ng.filter:currency)过滤器把一个数字格式化为金额的形式进行输出。

这个例子中最重要的一点是：Angular提供了**动态(live)**的绑定： 当 `input` 元素的值变化的时候，表达式的值也会自动重新计算，并且DOM所呈现的内容也会随着这些值的变化而自动更新。 这种模型(model)与视图(view)的联动就叫做“双向数据绑定(../tutorial/10.html)”。

添加UI逻辑：控制器

现在，我们添加一些逻辑，以便让这个例子支持不同的币种。它将允许我们使用不同的币种来输入、计算和支付这个订单。

源码

index.html | invoice1.js

```
<!doctype html>
<html ng-app="invoice1">
  <head>
    <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
    <script src="invoice1.js"></script>
  </head>
  <body>
    <div ng-controller="InvoiceController as invoice">
      <b>订单:</b>
      <div>
        数量: <input type="number" ng-model="invoice.qty" required >
      </div>
      <div>
        单价: <input type="number" ng-model="invoice.cost" required >
        <select ng-model="invoice.inCurr">
          <option ng-repeat="c in invoice.currencies"></option>
        </select>
      </div>
      <div>
        <b>总价:</b>
        <span ng-repeat="c in invoice.currencies">

        </span>
        <button class="btn" ng-click="invoice.pay()">支付</button>
      </div>
    </div>
  </body>
</html>
```

效果

Invoice:
Quantity: 1
Costs: 2 EUR
Total: USD2.70 EUR2.00 CNY16.46
Pay

这次有什么变动?

首先，出现了一个新的JavaScript文件，它包含一个被称为"控制器(controller) (../tutorial/2.html)"的函数。 更确切点说：这个文件中定义了一个构造函数，它用来在将来真正需要的时候创建这个控制器函数的实例。 控制器的用途是导出一些变量和函数，供模板中的表达式(expression)和指令(directive)使用。

在创建一个控制器的同时，我们还往HTML中添加了一个 `ng-controller` (api/ng.directive:ngController)指令。 这个指令告诉Angular，我们创建的这个 `InvoiceController` 控制器将会负责管理这个带有ng-controller指令的div节点，及其各级子节点。

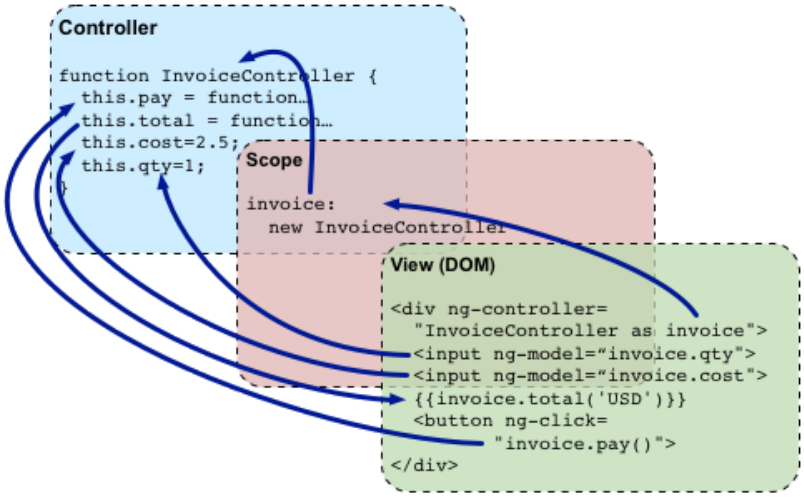
`InvoiceController as invoice` 这个语法告诉Angular：创建这个 `InvoiceController` 的实例，并且把这个实例赋值给当前作用域(Scope)中的 `invoice` 变量。

同时，我们修改了页面中所有用于读写Scope变量的表达式，给它们加上了一个 `invoice.` 前缀。 我们还把可选的币种作为一个数组定义在控制器中，并且通过 `ng-repeat` (api/ng.directive:ngRepeat)指令把它们添加到模板。 由于控制器中还包含了一个 `total` 函数，我们也能在DOM中使用 `{{ invoice.total(...) }}` 表达式来绑定总价的计算结果。

同样，这个绑定也是动态(live)的，也就是说：当invoice.total函数的返回值变化的时候，DOM也会跟着自动更新。 表单中的“支付”按钮附加上了指令 `ngClick` (api/ng.directive:ngClick)。 这意味着当它被点击时，会自动执行 `invoice.pay()` 这个表达式，即：调用当前作用域中的pay函数。

在这个新的JavaScript文件中，我们还创建了一个模块(module) (../tutorial/6.html)，并且在这个模块中注册了控制器(controller)。 接下来我们就讲一下模块(module)这个概念。

下面这个图表现的是我们声明了这个控制器(controller)之后，它们是如何协作的。



与视图(View)无关的业务逻辑：服务(Service)

现在， `InvoiceController` 包含了我们这个例子中的所有逻辑。如果这个应用程序的规模继续成长，最好的做法是：把控制器中与视图无关的逻辑都移到"服务(service) (guide/dev_guide.services)"中。 以便这个应用程序的其他部分也能复用这些逻辑。

接下来，就让我们重构我们的例子，并且把币种兑换的逻辑移入到一个独立的服务(service)中。

源码

index.html | finance2.js | invoice2.js

```
<!doctype html>
<html ng-app="invoice2">
  <head>
    <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
    <script src="finance2.js"></script>
    <script src="invoice2.js"></script>
  </head>
  <body>
    <div ng-controller="InvoiceController as invoice">
      <b>订单:</b>
      <div>
        数量: <input type="number" ng-model="invoice.qty" required >
      </div>
      <div>
        单价: <input type="number" ng-model="invoice.cost" required >
        <select ng-model="invoice.inCurr">
          <option ng-repeat="c in invoice.currencies"></option>
        </select>
      </div>
      <div>
        <b>总价:</b>
        <span ng-repeat="c in invoice.currencies">

        </span>
        <button class="btn" ng-click="invoice.pay()">支付</button>
      </div>
    </div>
  </body>
</html>
```

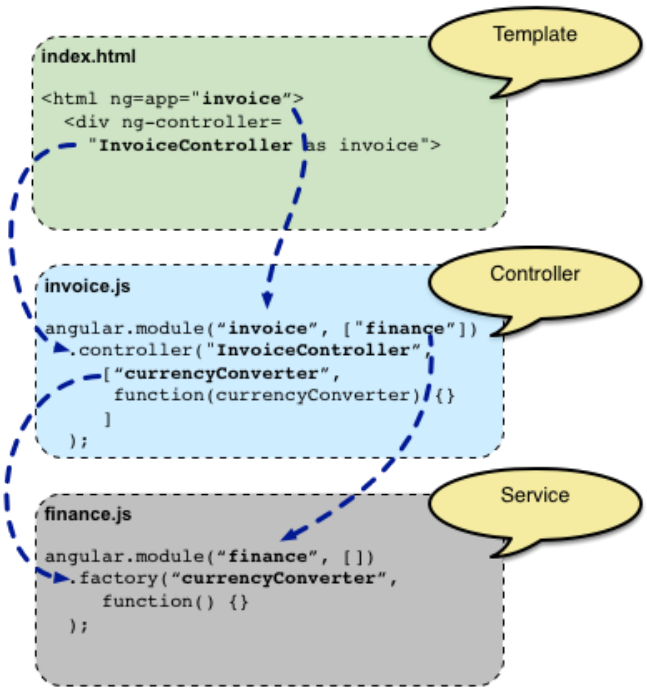
效果

Invoice:
Quantity: 1
Costs: 2 EUR
Total: USD2.70 EUR2.00 CNY16.46
Pay

这次有什么改动?

我们把 `convertCurrency` 函数和所支持的币种的定义独立到一个新的文件： `finance.js` 。但是控制器怎样才能找到这个独立的函数呢?

这下该“依赖注入(Dependency Injection) (../tutorial/17.html)”出场了。依赖注入(DI)是一种设计模式(Design Pattern)，它用于解决下列问题：我们创建了对象和函数，但是它们怎么得到自己所依赖的对象呢？ Angular中的每一样东西都是用依赖注入(DI)的方式来创建和使用的，比如指令(Directive)、过滤器(Filter)、控制器(Controller)、服务(Service)。 在Angular中，依赖注入(DI)的容器(container)叫做"注入器(injector) (../tutorial/17.html)"。



要想进行依赖注入，你必须先把这些需要协同工作的对象和函数注册(Register)到某个地方。在Angular中，这个地方叫做“模块(module)(../tutorial/6.html)”。

在上面这个例子中： 模板包含了一个 `ng-app="invoice2"` 指令。这告诉Angular使用 `invoice2` 模块作为该应用程序的主模块。 像 `angular.module('invoice', ['finance'])` 这样的代码告诉Angular： `invoice` 模块依赖于 `finance` 模块。 这样一来，Angular就能同时使用 `InvoiceController` 这个控制器和 `currencyConverter` 这个服务了。

刚才我们已经定义了应用程序的所有部分，现在，需要Angular来创建它们了。 在上一节，我们了解到控制器(controller)是通过一个工厂函数创建的。 而对于服务(service)，则有多种方式来定义它们的工厂函数（参见服务指南 (guide/dev_guide.services)）。 上面这个例子中，我们通过一个返回 `currencyConverter` 函数的函数作为创建 `currencyConverter` 服务的工厂。 （译注：js中的“工厂(factory)”是指一个以函数作为“返回值”的函数）

回到刚才的问题： `InvoiceController` 该怎样获得这个 `currencyConverter` 函数的引用呢？ 在Angular中，这非常简单，只要在构造函数中定义一些具有特定名字的参数就可以了。 这时，注入器(injector)就可以按照正确的依赖关系创建这些对象，并且根据名字把它们传入那些依赖它们的对象工厂中。 在我们的例子中， `InvoiceController` 有一个叫做 `currencyConverter` 的参数。 根据这个参数，Angular就知道 `InvoiceController` 依赖于 `currencyConverter` ，取得 `currencyConverter` 服务的实例，并且把它作为参数传给 `InvoiceController` 的构造函数。

这次改动中的最后一点是我们现在把一个数组作为参数传入到 `module.controller` 函数中，而不再是一个普通函数。 这个数组前面部分的元素包含这个控制器所依赖的一系列服务的名字，最后一个元素则是这个控制器的构造函数。 Angular可以通过这种数组语法来定义依赖，以免js代码压缩器(Minifier)破坏这个“依赖注入”的过程。 因为这些js代码压缩器通常都会把构造函数的参数重命名为很短的名

字，比如 a ，而常规的依赖注入是需要根据参数名来查找“被注入对象”的。 （译注：因为字符串不会被js代码压缩器重命名，所以数组语法可以解决这个问题。）

访问后端

现在开始最后一个改动：通过Yahoo Finance API来获得货币之间的当前汇率。 下面的例子将告诉你在Angular中应该怎么做。

源码

index.html | invoice3.js | finance3.js

```
<!doctype html>
<html ng-app="invoice3">
  <head>
    <script src="http://code.angularjs.org/1.2.25/angular.min.js"></script>
    <script src="invoice3.js"></script>
    <script src="finance3.js"></script>
  </head>
  <body>
    <div ng-controller="InvoiceController as invoice">
      <b>订单:</b>
      <div>
        数量: <input type="number" ng-model="invoice.qty" required >
      </div>
      <div>
        单价: <input type="number" ng-model="invoice.cost" required >
        <select ng-model="invoice.inCurr">
          <option ng-repeat="c in invoice.currencies"></option>
        </select>
      </div>
      <div>
        <b>总价:</b>
        <span ng-repeat="c in invoice.currencies">

        </span>
        <button class="btn" ng-click="invoice.pay()">Pay</button>
      </div>
    </div>
  </body>
</html>
```

效果

Invoice:
Quantity: 1
Costs: 2 EUR
Total:
Pay

这次有什么变动?

这次我们的 `finance` 模块中的 `currencyConverter` 服务使用了 `$http` (api/ng.$http)服务 —— 它是由Angular内建的用于访问后端API的服务。 是对 `XMLHttpRequest` (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest)以及JSONP (http://en.wikipedia.org/wiki/JSONP)的封装。详情参阅$http的API文档 - `$http` (../api/105.html)。