# 模板辅助器方法

相比辅助器方法，模板辅助器方法更智能一些，它们可以指定想要显示的属性，而让MVC框架去判断应该使用什么样的HTML元素。只是，需要一些初期关注才能建立起来，但毕竟是一种显示数据的更为灵活的方式。

这里打算继续使用介绍辅助器方法时使用的项目，但是，CreatePerson.cshtml视图在之前的辅助器方法会在生成的HTML元素上添加data属性，来支持表单验证，这一点在后面对模板辅助器方法的使用时打算禁用，但是，客户端验证特性对程序的其他部分仍然有效，调整后的代码如下（**粗体**部分为修改的内容）：

@model HelperMethods.Models.Person

@{

ViewBag.Title = "CreatePerson";

**Html.EnableClientValidation(false);**

}

<h2>CreatePerson</h2>

@using (Html.BeginRouteForm("FormRoute", new { }, FormMethod.Post,

new { @class = "personClass", data\_formType = "person" }))

{

<div class="dataElem">

<label>PersonId</label>

@Html.TextBoxFor(m => m.PersonId)

</div>

<div class="dataElem">

<label>First Name</label>

@Html.TextBoxFor(m => m.FirstName)

</div>

<div class="dataElem">

<label>Last Name</label>

@Html.TextBoxFor(m => m.LastName)

</div>

<div class="dataElem">

<label>Role</label>

@Html.DropDownListFor(m => m.Role, new SelectList(Enum.GetNames(typeof(HelperMethods.Models.Role))))

</div>

<input type="submit" value="提交" />

}

## 使用模板辅助器方法

首先来看看编辑元素的辅助器方法：Html.EditorFor和Html.Editor。Editor方法的字符串参数是用来指定编辑器元素所需的属性的。EditorFor是强类型的辅助器方法，可以使用lambda表达式指定编辑器元素所需要的模型属性。为了演示，下面同时混合使用了这两种方法，实际项目中可以根据自己的喜好使用，但是还是推荐使用EditorFor方法，这样可以避免由误输属性名造成的错误：

@model HelperMethods.Models.Person

@{

ViewBag.Title = "CreatePerson";

Html.EnableClientValidation(false);

}

<h2>CreatePerson</h2>

<!--使用模板辅助器方法-->

@using (Html.BeginRouteForm("FormRoute", new { }, FormMethod.Post, new { @class = "personClass", data\_formType = "person" }))

{

<div class="dataElem">

<label>PersonId</label>

@Html.Editor("PersonId")

</div>

<div class="dataElem">

<label>First Name</label>

@Html.Editor("FirstName")

</div>

<div class="dataElem">

<label>Last Name</label>

@Html.EditorFor(m => m.LastName)

</div>

<div class="dataElem">

<label>Role</label>

@Html.EditorFor(m => m.Role)

</div>

<div class="dataElem">

<label>Birth Date</label>

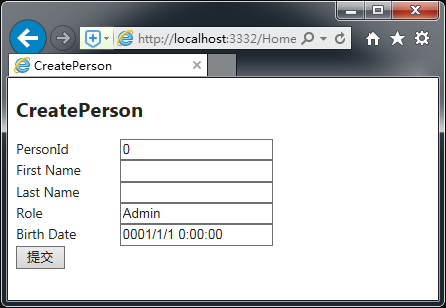
@Html.EditorFor(m => m.BirthDate)

</div>

<input type="submit" value="提交" />

}

最终效果如图，这和之前的辅助器方法实现的一样，只是增加了一个BirthDate的显示：



这里的日期显示成这个样子，是因为我的浏览器的问题，默认情况下IE浏览器对日期的显示不怎么友好，如果换成其他浏览器（如Opera浏览器）就可以正常显示，当然在IE下可以通过jQuery等一些第三方的控件实现。所以在开发Web端程序的时候一定要注意不同浏览器直接的差异（对于这几个属性的显示不同的还有PersonId等）。

HTML5规范对input元素所编辑的常规的属性类型，定义了一些不同的类型，如数字和日期，而这里的模板辅助器方法采用的就是HTML5的新类型。下面是我的浏览其中生成的HTML结果：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width">

<title>CreatePerson</title>

<link href="/Content/Site.css" rel="stylesheet">

<style type="text/css">

label {

display: inline-block;

width: 100px;

}

.dateElem {

margin: 5px;

}

</style>

</head>

<body>

<h2>CreatePerson</h2>

<!--使用模板辅助器方法-->

<form class="personClass" action="/app/forms/Home/CreatePerson" method="post" data-formtype="person"> <div class="dataElem">

<label>PersonId</label>

<input name="PersonId" class="text-box single-line" id="PersonId" **type="number"** value="0">

</div>

<div class="dataElem">

<label>First Name</label>

<input name="FirstName" class="text-box single-line" id="FirstName" **type="text"** value="">

</div>

<div class="dataElem">

<label>Last Name</label>

<input name="LastName" class="text-box single-line" id="LastName" **type="text"** value="">

</div>

<div class="dataElem">

<label>Role</label>

<input name="Role" class="text-box single-line" id="Role" **type="text"** value="Admin">

</div>

<div class="dataElem">

<label>Birth Date</label>

<input name="BirthDate" class="text-box single-line" id="BirthDate" **type="datetime"** value="0001/1/1 0:00:00">

</div>

<input type="submit" value="提交">

</form>

</body>

</html>

Type标签属性指定了input元素应当显示的类型，但是，可惜不是所有的浏览器都能支持，原因就是HTML5特性比较新。

后面会展示如何为MVC框架提供附加信息，以便改善辅助器方法生成的HTML，下面先看看完整的辅助器集。

|  |  |  |
| --- | --- | --- |
| 辅助器 | 示例 | 描述 |
| Display | Html.Display ("FirstName") | 渲染指定模型属性的只读视图，会根据该属性的类型及元数据选用一个HTML元素 |
| DisplayFor | Html.DisplayFor(x => x.FirstName) | 上一辅助器的强类型版本 |
| Editor | Html.Editor("FirstName") | 渲染指定模型属性的一个编辑器，会根据该属性的类型及元数据选用一个HTML元素 |
| EditorFor | Html.EditorFor(x => x.FirstName) | 上一辅助器的强类型版本 |
| Label | Html.Label("FirstName") | 渲染指定模型的HTML<label>元素（注意：它显示的是指定属性的属性名，而非属性值） |
| LabelFor | Html.LabelFor(x => x.FirstName) | 上一辅助器的强类型版本 |

### 生成标签和显示元素

后面对Home控制器做一下修改，用来演示这些辅助器方法：

using HelperMethods.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace HelperMethods.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

ViewBag.Fruits = new string[] { "Apple", "Orange", "Pear" };

ViewBag.Cities = new string[] { "New York", "London", "Paris" };

string message = "This is an HTML element: <input>";

return View((object)message);

}

public ActionResult CreatePerson()

{

return View(new Person());

}

[HttpPost]

public ActionResult CreatePerson(Person person)

{

**return View("DisplayPerson", person);**

}

}

}

上面的DisplayPerson对应的视图文件添加到/Views/Home文件夹中，其内容如下：

@model HelperMethods.Models.Person

@{

ViewBag.Title = "DisplayPerson";

}

<h2>DisplayPerson</h2>

<div class="dataElem">

@Html.Label("PersonId")

@Html.Display("PersonId")

</div>

<div class="dataElem">

@Html.Label("FirstName")

@Html.Display("FirstName")

</div>

<div class="dataElem">

@Html.LabelFor(m => m.LastName)

@Html.DisplayFor(m => m.LastName)

</div>

<div class="dataElem">

@Html.LabelFor(m => m.Role)

@Html.DisplayFor(m => m.Role)

</div>

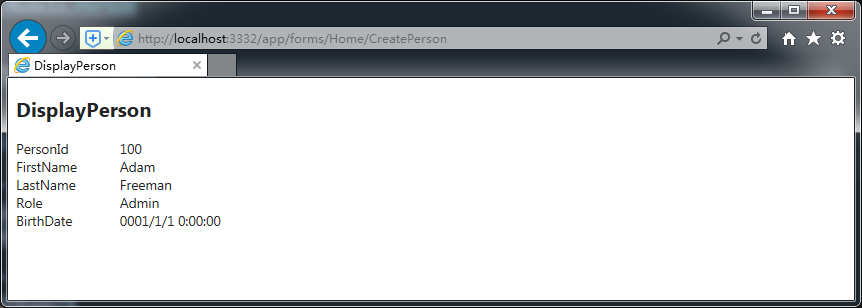
<div class="dataElem">

@Html.LabelFor(m => m.BirthDate)

@Html.DisplayFor(m => m.BirthDate)

</div>

启动程序，导航至/Home/CreatePerson后，填充内容，并点击“提交”按钮，便可看到下图的结果：



下面是这些辅助器方法生成的HTML，需要注意的是，Display和DisplayFor方法默认情况下并未生成HTML元素——它们只是发布了它们所操作的属性值：

<!DOCTYPE html>

<html><head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width">

<title>DisplayPerson</title>

<link href="/Content/Site.css" rel="stylesheet">

<style type="text/css">

label {

display: inline-block;

width: 100px;

}

.dateElem {

margin: 5px;

}

</style>

</head>

<body>

<h2>DisplayPerson</h2>

<div class="dataElem">

**<label for="PersonId">PersonId</label>**

**100**

</div>

<div class="dataElem">

**<label for="FirstName">FirstName</label>**

**Adam**

</div>

<div class="dataElem">

**<label for="LastName">LastName</label>**

**Freeman**

</div>

<div class="dataElem">

**<label for="Role">Role</label>**

**Admin**

</div>

<div class="dataElem">

**<label for="BirthDate">BirthDate</label>**

**0001/1/1 0:00:00**

</div>

</body></html>

看似这些辅助器不是很有用，但可以做些调整来使其产生更希望显示给用户的输出结果。

### 使用整体模型模板辅助器

前面说的都是针对单个属性的辅助器，其实还有一些针对整个对象的辅助器——这称之为**支架辅助器。**

|  |  |  |
| --- | --- | --- |
| 辅助器 | 示例 | 描述 |
| DisplayForModel | Html.DisplayForModel() | 渲染整个模型对象的只读视图 |
| EditorForModel | Html.EditorForModel() | 渲染整个模型对象的编辑器元素 |
| LabelForModel | Html.LabelForModel() | 渲染对整个模型对象进行引用的HTML<label>元素 |

注：所谓**模板辅助器**（Templated Helper）是在视图中使用的一种辅助性工具，用以为视图模型的属性生成HTML标记。模板辅助器有两大类：一类就叫**模板辅助器**（Templated Helper），作用是在视图中为模型的个别属性生成HTML标记。另一类叫作**支架辅助器**（Scaffolding Helper），作用是在视图中创建整个模型所有属性的HTML标记。这两种模板辅助器又分别有三种：

* 标签辅助器（Label、LabelFor、LabelForModel），用于生成模型属性的标签（以属性名称作为标签）；
* 显示辅助器（Display、DisplayFor、DisplayForModel），用于生成模型属性的显示标记（显示属性的值，或显示只读数据）；
* 编辑辅助器（Edit、EditFor、EditForModel），用于生成对模型属性进行编辑的标记（编辑器）。

下面演示一下通过LabelForModel和EditForModel 辅助器简化CreatePerson.cshtml视图的结果：

@model HelperMethods.Models.Person

@{

ViewBag.Title = "CreatePerson";

Html.EnableClientValidation(false);

}

<h2>CreatePerson: **@Html.LabelForModel()**</h2>

@using (Html.BeginRouteForm("FormRoute", new { }, FormMethod.Post, new { @class = "personClass", data\_formType = "person" }))

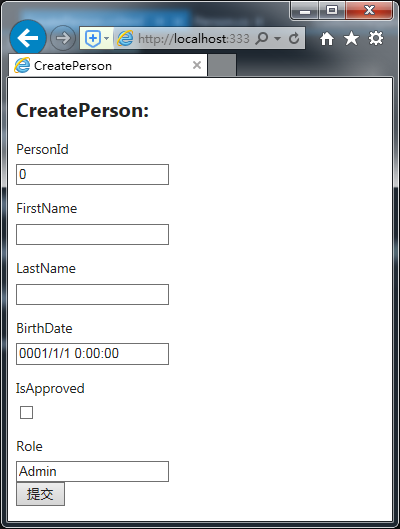
{

**@Html.EditorForModel()**

<input type="submit" value="提交" />

}

下面是效果，虽然还有些不太正确，主要是因为支架辅助器生成的HTML与前面的布局视图中CSS的定义不对应，以及它会默认显示所有的模型中的属性，后面做一些修改便可达到我们的目标：



下面先通过对样式作简单的改动，对视图外观加以整理，以使它们预制件辅助器添加到div和input元素中的CSS的class值对应起来。文件\_Layout.cshtml修改的结果：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

<link href="~/Content/Site.css" rel="stylesheet" />

<style type="text/css">

label {

display: inline-block;

width: 100px;

}

.dateElem {

margin: 5px;

}

**h2 > label {**

**width: inherit;**

**}**

**.editor-label, .editor-field {**

**float: left;**

**}**

**.editor-label, .editor-label label, .editor-field input {**

**height: 20px;**

**}**

**.editor-label {**

**clear: left;**

**}**

**.editor-field {**

**margin-left: 10px;**

**margin-top: 10px;**

**}**

**input[type=submit] {**

**float: left;**

**clear: both;**

**margin-top: 10px;**

**}**

**.column {**

**float: left;**

**margin: 10px;**

**}**

</style>

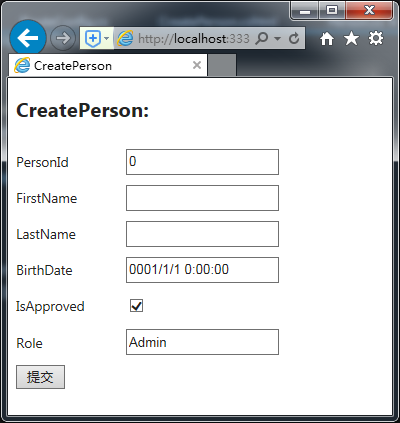
</head>

<body>

@RenderBody()

</body>

</html>



## 使用模型元数据

用前面的辅助器模板生成的HTML可以看出其并非十分智能，经常会产生不需要的HTML。但是，我们可以通过**模型元数据（MetaData）**加以辅助，从而改善这种结果。它的实现原理是：使用C#注解属性，以及注解属性的参数值来为辅助器提供一定的指示，辅助器方法在生成HTML元素时将会根据这些信息进行组织。

### 用元数据控制编辑及可见性

比如我们通过HiddenInput注解属性将Person类中的PersonId属性隐藏后，在运用这个注解属性时，Html.EditorFor和Html.EditorForModel辅助器会生成一个对应的只读视图，下面分别是代码示例和效果图：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

**using System.Web.Mvc;**

namespace HelperMethods.Models

{

public class Person

{

**[HiddenInput]**

public int PersonId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime BirthDate { get; set; }

public Address HomeAddress { get; set; }

public bool IsApproved { get; set; }

public Role Role { get; set; }

}

public class Address

{

public string Line1 { get; set; }

public string Line2 { get; set; }

public string City { get; set; }

public string PostalCode { get; set; }

public string Country { get; set; }

}

public enum Role

{

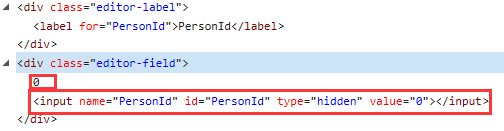
Admin,

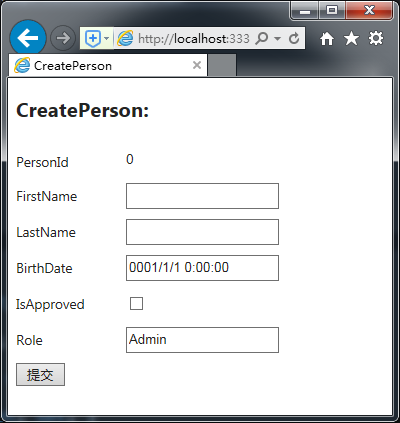
User,

Guest

}

}





这样得到了一个该属性值的文本显示效果，且在HTML中生成了一个字符串值和一个隐藏的input元素。如果不希望显示该属性值，可以将该注解属性的DisplayValue参数设置为false，如：

…

public class Person

{

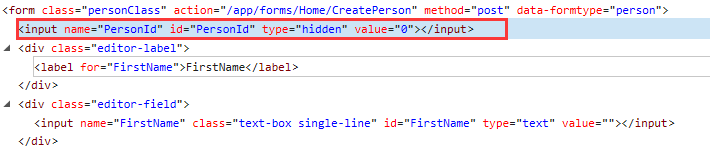
**[HiddenInput(DisplayValue = false)]**

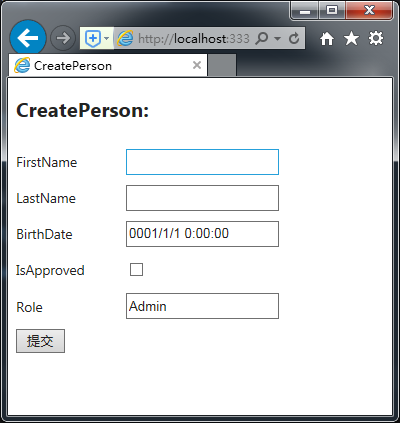
public int PersonId { get; set; }

…

}

…





此时，即便使用的是类似Html.EditorFor这样的针对某一属性的模板辅助器，任会达到上面的效果。

排除支架中的一个属性

如果不需要某一属性生成HTML元素，可以使用ScaffoldColumn（命名空间为：System.ComponentModel.DataAnnotations）注解属性。相对于HiddenInput，ScaffoldColumn是将某一属性标记为完全禁止，即在生成HTML的时候将不会将该属性生成对应的HTML元素，如下面示例：

**[ScaffoldColumn(false)]**

public int PersonId { get; set; }

但是，这么做也有一个问题，就是这对模型绑定是有影响的，而且对模板辅助器（相对于支架辅助器的那种）是不起作用的。

### 使用用于标签的元数据

默认情况下，Label、LabelFor、LabelForModel，以及EditorForMordel辅助器会以属性名称作为它们生成的标签元素的内容，但实际项目中经常不希望这么做，尤其是在中文环境的项目中。这个时候可以使用System.ComponentModel命名空间下的DisplayName注解属性，并为其参数提供希望的值即可（同时还有System.ComponentModel.DataAnnotations中的Display注解属性）。如：

using System;

using System.Collections.Generic;

**using System.ComponentModel;**

**using System.ComponentModel.DataAnnotations;**

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace HelperMethods.Models

{

// DisplayName 也可以用于属性，但习惯上建议在模型类上应用，而在属性上使用 Display

**[DisplayName("New Person")]**

public class Person

{

[HiddenInput(DisplayValue = false)]

public int PersonId { get; set; }

**[Display(Name = "First")]**

public string FirstName { get; set; }

**[Display(Name = "Last")]**

public string LastName { get; set; }

**[Display(Name = "Birth Date")]**

public DateTime BirthDate { get; set; }

public Address HomeAddress { get; set; }

**[Display(Name = "Approved")]**

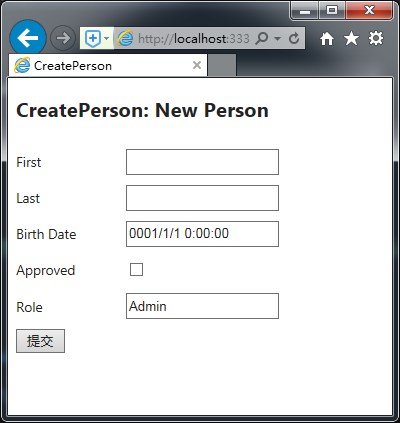
public bool IsApproved { get; set; }

public Role Role { get; set; }

}

…

}



### 使用用于数据值的元数据

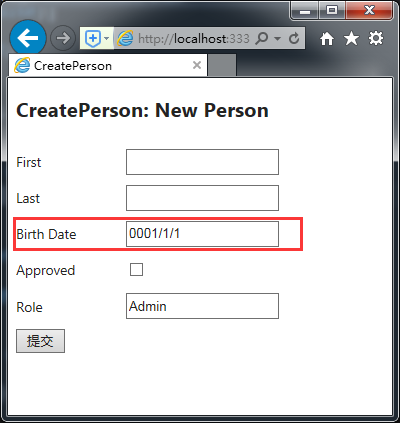
可以用元数据为如何显示模型属性提供一些提示。如，如果需要DateTime类型的属性值显示为日期格式，不需要时间，此时便可以使用DataType注解属性。如下示例：

[Display(Name = "Birth Date")]

**[DataType(DataType.Date)]**

public DateTime BirthDate { get; set; }

效果如图：



DataType注解属性的参数DataType参数是一个枚举类型的对象，其中最常用的一些枚举值如下所列：

* DateTime：显示日期和时间
* Date：显示DateTime的日期部分
* Time：显示DateTime的时间部分
* Text：显示单行文本
* PhoneNumber：显示电话号码
* MultilineText：将值渲染在一个文本区域（textarea）元素中（多行文本）

注意：该值会让编辑器辅助器创建一个HTML的textarea元素，但显示辅助器对这个值是忽略的——textarea的作用是让用户编辑一个值，但在一个只读表单中显示数据时，textarea就没有意义了。

* Password：表示一个密码值
* Url：将数据显示为一个URL（用HTML的a元素，Url的值仅对显示辅助器起作用）
* EmailAddress：将数据显示为一个E-mail地址（使用带有mailto的href的a元素）

### 用元数据选择显示模板

模板辅助器都是使用显示模板来生成HTML的。辅助器使用的模板是基于所处理的书写的类型以及辅助器的种类的。要是希望使用指定的模板处理属性，则可以使用UIHint注解属性。

如下面的演示：

…

[Display(Name = "First")]

**[UIHint("MultilineText")]**

public string FirstName { get; set; }

…

下面是一些内建的辅助器模板：

|  |  |  |
| --- | --- | --- |
| 值（模板） | 效果（编辑器辅助器） | 效果（显示辅助器） |
| Boolean | 渲染一个bool值的复选框。对于nullable的bool?值，创建一个带有True、False和NotSet选项的select元素 | 与编辑器辅助器相同，但附加了disabled标签属性，以渲染只读HTML控件 |
| Collection | 为IEnumerable序列中的每一个元素渲染一个相应的模板，该序列中的各个项不必是同一类型 | 与编辑器辅助器相同 |
| Decimal | 渲染一个单行文本框的input元素，并对数据值格式化，显示两位小数 | 渲染格式化两位小数的数据值 |
| DateTime | 渲染一个input元素，其type标签属性为datetime，并且包含完整的日期和时间 | 渲染DateTime变量的完整值 |
| Date | 渲染一个input元素，其type标签属性为date，只包含日期成份（不包含时间） | 渲染DateTime变量的日期部分 |
| EmailAddress | 将值渲染在一个单行文本框的input元素 | 用HTML的a元素渲染一个链接，且href标签属性格式化成一个mailto的URL |
| HiddenInput | 创建隐藏的input元素 | 渲染该数据值，并创建一个隐藏的input元素 |
| Html | 将值渲染在一个单行文本框的input元素 | 用HTML的a元素渲染一个链接 |
| MultilineText | 渲染一个含有该数据值的HTML的textarea元素 | 渲染数据值 |
| Number | 渲染一个input元素，其type标签属性被设置为number | 渲染数据值 |
| Password | 将值渲染在一个单行文本框的input元素中，使字符不显示，但可以编辑 | 渲染数据值——字符是非隐藏的（明文显示） |
| String | 将值渲染在单行文本框的input元素中 | 渲染数据值 |
| Text | 等同于String模板 | 等同于String模板 |
| Tel | 渲染一个input元素，其type标签属性被设置为tel | 渲染数据值 |
| Time | 渲染一个input元素，其type标签属性为time，只包含时间部分 | 渲染DateTime变量的时间部分 |
| Url | 将值渲染在单行文本框的input元素中 | 用HTML的a元素渲染一个链接。HTML内文本和href标签属性都设置为该数据值。 |

注意：在使用UIHint注解属性时必须小心。如果UIHint运用于一个属性，但选择了一个不能对该属性的类型进行操作的模板，将会报出异常的。

Object模板是一个特殊的情况——它是由支架辅助器用来为一个视图模型对象生成HTML的模板。这个模板会检查对象的每一个属性，并为相应的属性类型选择最合适的模板。Object模板会把注入UIHint以及DataType之类的元数据考虑进去。