# 模型验证

**模型验证**（Model Validation）：是确保用户接收的数据适合于绑定的模型，并且在不合适时，给用户提供有用的信息，以帮助他们修正其问题的过程。

**模型验证过程一**：检查接收的数据——是保持域模型完整性的方式之一。

**模型验证过程二**：帮助用户修正问题。

## 示例项目介绍

项目模板：Basic

项目名称：ModelValidation

一个新的模型类文件：Appointment.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace ModelValidation.Models

{

public class Appointment

{

public string ClinetName { get; set; }

[DataType(DataType.Date)]

public DateTime Date { get; set; }

public bool TermsAccepted { get; set; }

}

}

控制器：Home，用来提供处理Appointment模型类的动作方法

using ModelValidation.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ModelValidation.Controllers

{

public class HomeController : Controller

{

public ViewResult MakeBooking()

{

return View(new Appointment { Date = DateTime.Now });

}

[HttpPost]

public ViewResult MakeBooking(Appointment appt)

{

// 在实际项目中，此处是存储库中存储新的 Appointment 的语句

return View("Completed", appt);

}

}

}

本例不打算实现具体存储等方面的功能，主要是为了能够将注意力集中在模型绑定以及验证过程方面。要清楚验证模型的主要目的就为了防止在存储库中放置劣质或无意义的数据，并避免引起问题。

下面继续创建两个视图MakeBooking.cshtml、Completed.cshtml：

MakeBooking.cshtml，用来提供一个让用户创建新预约的表单：

@model ModelValidation.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

}

<h4>Book an Appointment</h4>

@using (Html.BeginForm())

{

<p>Your name: @Html.EditorFor(m => m.ClinetName)</p>

<p>Appointment Date: @Html.EditorFor(m => m.Date)</p>

<p>@Html.EditorFor(m => m.TermsAccepted) I accept the terms $ conditions</p>

<input type="submit" value="Make Booking" />

}

用于提交表单后给用户显示预约细节的内容的Completed.cshtml视图：

@model ModelValidation.Models.Appointment

@{

ViewBag.Title = "Completed";

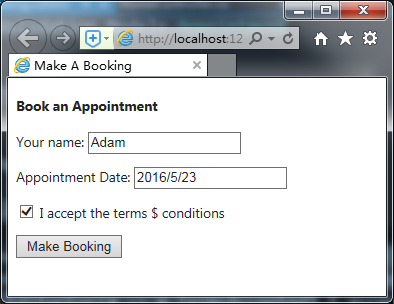
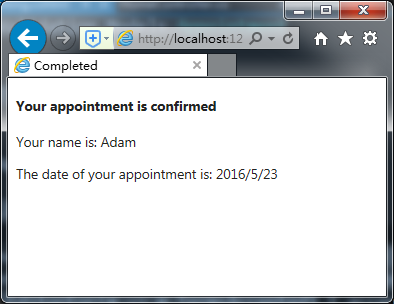
}

<h4>Your appointment is confirmed</h4>

<p>Your name is: @Html.DisplayFor(m => m.ClinetName)</p>

<p>The date of your appointment is: @Html.DisplayFor(m => m.Date)</p>

导航地址到/Home/MakeBooking可以看到效果：

按照现在的情况，此刻程序会接收用户递交的任何数据，但为了保持程序和域模型的完整性，在接收用户递交的Appointment之前，需要确保三件事为“true”。

* 用户必须提供一个姓名
* 用户必须提供一个未来的日期（以yyyy/mm/dd格式）
* 用户必须选择了复选框，已接受条款和条件

在这个示例中，模型验证就是实施这些需求的过程。后面将采不同的技术来实现，以检查用户提供的数据，并在输入无效数据时给予提示。

## 明确地验证模型

最直接的验证模型的方式是在动作方法中进行验证。下面是在MakeBooking动作方法的HttpPost版本中对Appointment类所定义的每一个属性添加明确的检查：

[HttpPost]

public ViewResult MakeBooking(Appointment appt)

{

**if (string.IsNullOrEmpty(appt.ClinetName))**

**{**

**ModelState.AddModelError("ClinetName", "Please enter your name");**

**}**

**if (ModelState.IsValidField("Date") && DateTime.Now > appt.Date)**

**{**

**ModelState.AddModelError("Date", "Please enter a date in the future");**

**}**

**if (!appt.TermsAccepted)**

**{**

**ModelState.AddModelError("TermsAccepted", "You must accept the terms");**

**}**

**if (ModelState.IsValid)**

**{**

// 在实际项目中，此处是存储库中存储新的 Appointment 的语句

return View("Completed", appt);

**}**

**else**

**{**

**return View();**

**}**

}

上面对Appointment对象的各个属性进行检测时，使用的ModelState.IsValidField方法检查的是绑定器是否能够对一个属性赋值。

注意：如果无法从请求数据中解析到一个值，执行额外检测或报告其他错误消息是没有意义的。

上面示例中在验证了模型对象的所有属性之后，读取了ModelState.IsValid属性，以考察是否有错误发送，如果在检查期间调用Model.State.AddModelError方法，或创建Appointment对象时模型绑定器遇到了问题，该方法返回false。

if (ModelState.IsValid)

{

// 在实际项目中，此处是存储库中存储新的 Appointment 的语句

return View("Completed", appt);

}

else

{

return View();

}

如果IsValid值为true，则会得到一个有效的Appointment对象，且会渲染Completed.cshtml视图；如果是false，则说明出现了问题，此时将掉牙View方法渲染默认视图。

**向用户显示验证错误**

通过调用View方法来处理验证错误看上去可能有点怪，但是在MakeBooking.cshtml视图中用来生成input元素的模板视图辅助器，会检查视图模型的验证错误。

如果一个属性报告了错误，那么辅助器对相应的input元素添加一个CSS的class标签属性，其值为input-validation-error**。**~/Content/site.css文件中包含了一个用于该样式的默认定义，如：

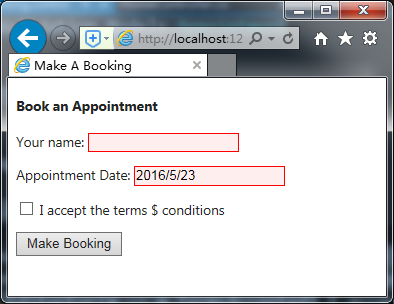
.input-validation-error {

border: 1px solid #f00;

background-color: #fee;

}

验证错误的效果如图：



设置复选框样式

有些浏览器，包括Chrome和Firefox，会忽略运用于复选框的样式，这会导致不协调的视觉反馈。解决办法是在~/Views/Shared/EditorTemplates/Boolean.cshtml中创建一个自定义模板，并将复选框封装在一个div元素中。以下是所使用的一个模板，但我们可以将其定制到自己的应用程序中：

@model bool?

@if (ViewData.ModelMetadata.IsNullableValueType)

{

@Html.DropDownListFor(m => m, new SelectList(new[] { "Not Set", "True", "False" }, Model))

}

else

{

ModelState state = ViewData.ModelState[ViewData.ModelMetadata.PropertyName];

bool value = Model ?? false;

if (state != null && state.Errors.Count > 0)

{

<div class="input-validation-error" style="float:left">

@Html.CheckBox("", value)

</div>

}

else

{

@Html.CheckBox("", value)

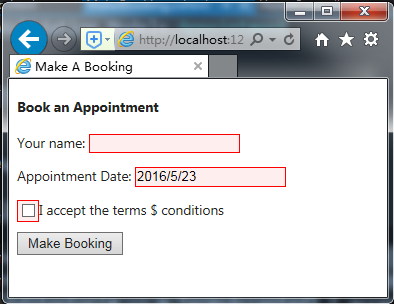
}

}

如果运用这个模板的属性出现错误，该模板会将一个复选框（CheckBox）封装在一个运用了input-validation-error样式的div元素中。

只有当提交的数据能被模型绑定器解析，且能被验证通过，才会显示完成视图，否则，将会提示验证失败的视图。

运用了上面复选框样式后的效果：



## 显示验证消息

## 使用其他验证技术

### 在模型绑定器中执行验证

### 用元数据指定验证规则

1. 创建自定义的属性验证注解属性
2. 创建模型验证注解属性

### 定义自验证模型

## 执行客户端验证

### 启用和禁用客户端验证

### 使用客户端验证

### 理解客户端验证机制

## 执行远程验证