عنوان: ASP.NET MVC #10
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۷ ۰۰:۳۰:۰۰
آدرس: www.dotnettips.info
گروه‌ها: MVC

آشنایی با روش‌های مختلف ارسال اطلاعات یک درخواست به کنترلر

تا اینجا با روش‌های مختلف ارسال اطلاعات از یک کنترلر به View متناظر آن آشنا شدیم. اما حالت عکس آن چطور؟ مثلا در ASP.NET Web forms، دوبار بر روی یک دکمه کلیک می‌کردیم و در روال رویدادگردان کلیک آن، همانند برنامه‌های ویندوزی، دسترسی به اطلاعات اشیاء قرار گرفته بر روی فرم را داشتیم. در ASP.NET MVC که کلا مفهوم Events را حذف کرده و وب را همانگونه که هست ارائه می‌دهد و به علاوه کنترلرهای آن، ارجاع مستقیمی را به هیچکدام از اشیاء بصری در خود ندارند (برای مثال کنترلر و متدی در آن نمی‌دانند که الان بر روی View آن، یک گرید قرار دارد یا یک دکمه یا اصلا هیچی)، چگونه می‌توان اطلاعاتی را از کاربر دریافت کرد؟

در اینجا حداقل سه روش برای دریافت اطلاعات از کاربر وجود دارد:

الف) استفاده از اشیاء Context مانند HttpContext، Request، RouteData و غیره

ب) به کمک پارامترهای اکشن متدها

ج) با استفاده از ویژگی جدیدی به نام Data Model Binding

یک مثال کاربردی

قصد داریم یک صفحه لاگین ساده را طراحی کنیم تا بتوانیم هر سه حالت ذکر شده فوق را در عمل بررسی نمائیم. بحث HTML Helpers استاندارد ASP.NET MVC را هم که در قسمت قبل شروع کردیم، لابلای توضیحات قسمت جاری و قسمت‌های بعدی با مثال‌های کاربردی دنبال خواهند شد.

بنابراین یک پروژه جدید خالی ASP.NET MVC را شروع کرده و مدلی را به نام Account با محتوای زیر به پوشه Models برنامه اضافه کنید:

```
namespace MvcApplication6.Models
{
    public class Account
    {
        public string Name { get; set; }
        public string Password { get; set; }
    }
}
```

یک کنترلر جدید را هم به نام LoginController به پوشه کنترلرهای برنامه اضافه کنید. بر روی متد Index پیش فرض آن کلیک راست نمائید و یک View خالی را اضافه نمائید.

در ادامه به فایل Global.asax.cs مراجعه کرده و نام کنترلر پیش‌فرض را به Login تغییر دهید تا به محض شروع برنامه در VS.NET، صفحه لاگین ظاهر شود.

کدهای کامل کنترلر لاگین را در ادامه ملاحظه می‌کنید:

```
using System.Web.Mvc;
using MvcApplication6.Models;

namespace MvcApplication6.Controllers
{
    public class LoginController : Controller
    {
        [HttpGet]
        public ActionResult Index()
```

```
        {
            return View(); //Shows the login page
        }

        [HttpPost]
        public ActionResult LoginResult()
        {
            string name = Request.Form["name"];
            string password = Request.Form["password"];

            if (name == "Vahid" && password == "123")
                ViewBag.Message = "Succeeded";
            else
                ViewBag.Message = "Failed";

            return View("Result");
        }

        [HttpPost]
        [ActionName("LoginResultWithParams")]
        public ActionResult LoginResult(string name, string password)
        {
            if (name == "Vahid" && password == "123")
                ViewBag.Message = "Succeeded";
            else
                ViewBag.Message = "Failed";

            return View("Result");
        }

        [HttpPost]
        public ActionResult Login(Account account)
        {
            if (account.Name == "Vahid" && account.Password == "123")
                ViewBag.Message = "Succeeded";
            else
                ViewBag.Message = "Failed";

            return View("Result");
        }
    }
}
```

همچنین Viewهای متناظر با این کنترلر هم به شرح زیر هستند:

فایل index.cshtml به نحو زیر تعریف خواهد شد:

```
@model MvcApplication6.Models.Account
@{
    ViewBag.Title = "Index";
}
<h2>
    Login</h2>
@using (Html.BeginForm(actionName: "LoginResult", controllerName: "Login"))
{
    <fieldset>
        <legend>Test LoginResult()</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)</p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)</p>
        <input type="submit" value="Login" />
    </fieldset>
}
@using (Html.BeginForm(actionName: "LoginResultWithParams", controllerName: "Login"))
{
    <fieldset>
        <legend>Test LoginResult(string name, string password)</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)</p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)</p>
        <input type="submit" value="Login" />
    </fieldset>
}
```

```
@using (Html.BeginForm(actionName: "Login", controllerName: "Login"))
{
    <fieldset>
        <legend>Test Login(Account acc)</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)</p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)</p>
        <input type="submit" value="Login" />
    </fieldset>
}
```

و فایل result.cshtml هم محتوای زیر را دارد:

```
@{
    ViewBag.Title = "Result";
}
<fieldset>
    <legend>Login Result</legend>
    <p>
        @ViewBag.Message</p>
</fieldset>
```

توضیحاتی در مورد View لاگین برنامه:

در View صفحه لاگین سه فرم را مشاهده می‌کنید. در برنامه‌های ASP.NET Web forms در هر صفحه، تنها یک فرم را می‌توان تعریف کرد؛ اما در ASP.NET MVC این محدودیت برداشته شده است.

تعریف یک فرم هم با متد کمکی Html.BeginForm انجام می‌شود. در اینجا برای مثال می‌شود یک فرم را به کنترلری خاص و متدی مشخص در آن نگاشت نمائیم.

از عبارت using هم برای درج خودکار تگ بسته شدن فرم، در حین dispose شیء MvcForm کمک گرفته شده است.

برای نمونه خروجی HTML اولین فرم تعریف شده به صورت زیر است:

```
<form action="/Login/LoginResult" method="post">
    <fieldset>
        <legend>Test LoginResult()</legend>
        <p>
            Name: <input id="Name" name="Name" type="text" value="" /></p>
        <p>
            Password: <input id="Password" name="Password" type="password" /></p>
        <input type="submit" value="Login" />
    </fieldset>
</form>
```

توسط متدهای کمکی Html.TextBoxFor و Html.PasswordFor یک TextBox و یک PasswordBox به صفحه اضافه می‌شوند، اما این For آن‌ها و همچنین lambda expression ایی که بکارگرفته شده برای چیست؟

متدهای کمکی Html.TextBox و Html.Password از نگارش‌های اولیه ASP.NET MVC وجود داشتند. این متدها نام خاصیت‌ها و پارامترهایی را که قرار است به آن‌ها بایند شوند، به صورت رشته می‌پذیرند. اما با توجه به اینکه در اینجا می‌توان یک strongly typed view را تعریف کرد، تیم ASP.NET MVC بهتر دیده است که این رشته‌ها را حذف کرده و از قابلیتی به نام Static reflection استفاده کند (^ و ^).

با این توضیحات، اطلاعات سه فرم تعریف شده در View لاگین برنامه، به سه متد متفاوت قرار گرفته در کنترلری به نام Login ارسال خواهند شد. همچنین با توجه به مشخص بودن نوع model که در ابتدای فایل تعریف شده، خاصیت‌هایی را که قرار است اطلاعات ارسالی به آن‌ها بایند شوند نیز به نحو strongly typed تعریف شده‌اند و تحت نظر کامپایلر خواهند بود.

توضیحاتی در مورد نحوه عملکرد کنترلر لاگین برنامه:

در این کنترلر صرفنظر از محتوای متدهای آن‌ها، دو نکته جدید را می‌توان مشاهده کرد. استفاده از ویژگی‌های HttpGet، HttpPost و ActionName. در اینجا به کمک ویژگی‌های HttpGet و HttpPost در مورد نحوه دسترسی به این متدها، محدودیت قائل شده‌ایم. به این معنا که تنها در حالت Post است که متد LoginResult در دسترس خواهد بود و اگر شخصی نام این متدها را مستقیما در مرورگر وارد کند (یا همان HttpGet پیش فرض که نیازی هم به ذکر صریح آن نیست)، با پیغام «یافت نشد» مواجه می‌گردد.
البته در نگارش‌های اولیه ASP.NET MVC از ویژگی دیگری به نام AcceptVerbs برای مشخص سازی نوع محدودیت فراخوانی یک اکشن متد استفاده می‌شد که هنوز هم معتبر است. برای مثال:

```
[AcceptVerbs(HttpVerbs.Get)]
```

یک نکته امنیتی:
همیشه متدهای Delete خود را به HttpPost محدود کنید. به این علت که ممکن است در طی مثلا یک ایمیل، آدرسی به شکل http://localhost/blog/delete/10 برای شما ارسال شود و همچنین سشن کار با قسمت مدیریتی بلاگ شما نیز در همان حال فعال باشد. URL ایی به این شکل، در حالت پیش فرض، محدودیت اجرایی HttpGet را دارد. بنابراین احتمال اجرا شدن آن بالا است. اما زمانیکه متد delete را به HttpPost محدود کردید، دیگر این نوع حملات جواب نخواهند داد و حتما نیاز خواهد بود تا اطلاعاتی به سرور Post شود و نه یک Get ساده (مثلا کلیک بر روی یک لینک معمولی)، کار حذف را انجام دهد.

توسط ActionName می‌توان نام دیگری را صرفنظر از نام متد تعریف شده در کنترلر، به آن متد انتساب داد که توسط فریم ورک در حین پردازش نهایی مورد استفاده قرار خواهد گرفت. برای مثال در اینجا به متد LoginResult دوم، نام LoginResultWithParams را انتساب داده‌ایم که در فرم دوم تعریف شده در View لاگین برنامه مورد استفاده قرار گرفته است.
وجود این ActionName هم در مثال فوق ضروری است. از آنجائیکه دو متد هم نام را معرفی کرده‌ایم و فریم ورک نمی‌داند که کدامیک را باید پردازش کند. در این حالت (بدون وجود ActionName معرفی شده)، برنامه با خطای زیر مواجه می‌گردد:

```
The current request for action 'LoginResult' on controller type 'LoginController' is ambiguous between
the following action methods:
System.Web.Mvc.ActionResult LoginResult() on type MvcApplication6.Controllers.LoginController
System.Web.Mvc.ActionResult LoginResult(System.String, System.String) on type
MvcApplication6.Controllers.LoginController
```

برای اینکه بتوانید نحوه نگاشت فرم‌ها به متدها را بهتر درک کنید، بر روی چهار return View موجود در کنترلر لاگین برنامه، چهار breakpoint را تعریف کنید. سپس برنامه را در حالت دیباگ اجرا نمائید و تک تک فرم‌ها را یکبار با کلیک بر روی دکمه لاگین، به سرور ارسال نمائید.

بررسی سه روش دریافت اطلاعات از کاربر در ASP.NET MVC

الف) استفاده از اشیاء Context

در ویژوال استودیو، در کنترلر لاگین برنامه، بر روی کلمه Controller کلیک راست کرده و گزینه Go to definition را انتخاب کنید. در اینجا بهتر می‌توان به خواصی که در یک کنترلر به آن‌ها دسترسی داریم، نگاهی انداخت:

```
public HttpContextBase HttpContext { get; }
public HttpRequestBase Request { get; }
```

```
public HttpResponseBase Response { get; }
public RouteData RouteData { get; }
```

در بین این خواص و اشیاء مهیا، Request و RouteData بیشتر مد نظر ما هستند. در مورد RouteData در قسمت ششم این سری، توضیحاتی ارائه شد. اگر مجددا Go to definition مربوط به HttpRequestBase خاصیت Request را بررسی کنیم، موارد ذیل جالب توجه خواهند بود:

```
public virtual NameValueCollection QueryString { get; } // GET variables
public NameValueCollection Form { get; } // POST variables
public HttpCookieCollection Cookies { get; }
public NameValueCollection Headers { get; }
public string HttpMethod { get; }
```

توسط خاصیت Form شیء Request می‌توان به مقادیر ارسالی به سرور در یک کنترلر دسترسی یافت که نمونه‌ای از آن‌را در اولین متد LoginResult می‌توانید مشاهده کنید. این روش در ASP.NET Web forms هم کار می‌کند. جهت اطلاع این روش با ASP کلاسیک دهه نود هم سازگار است!

البته این روش آنچنان مرسوم نیست؛ چون NameValueCollection مورد استفاده، ایندکسی عددی یا رشته‌ای را می‌پذیرد که هر دو با پیشرفت‌هایی که در زبان‌های دات نتی صورت گرفته‌اند، دیگر آنچنان مطلوب و روش مرجح به حساب نمی‌آیند. اما ... هنوز هم قابل استفاده است.

به علاوه اگر دقت کرده باشید در اینجا HttpContextBase داریم بجای HttpContext. تمام این کلاس‌های پایه هم به جهت سهولت انجام آزمون‌های واحد در ASP.NET MVC ایجاد شده‌اند. کار کردن مستقیم با HttpContext مشکل بوده و نیاز به شبیه سازی فرآیندهای رخ داده در یک وب سرور را دارد. اما این کلاس‌های پایه جدید، مشکلات یاد شده را به همراه ندارند.

ب) استفاده از پارامترهای اکشن متدها

نکته‌ای در مورد نامگذاری پارامترهای یک اکشن متد به صورت توکار اعمال می‌شود که باید به آن دقت داشت:

اگر نام یک پارامتر، با نام کلید یکی از رکوردهای موجود در مجموعه‌های زیر یکی باشد، آنگاه به صورت خودکار اطلاعات دریافتی به این پارامتر نگاشت خواهد شد (پارامتر هم نام، به صورت خودکار مقدار دهی می‌شود). این مجموعه‌ها شامل موارد زیرهستند:

```
Request.Form
Request.QueryString
Request.Files
RouteData.Values
```

برای نمونه در متدی که با نام LoginResultWithParams مشخص شده، چون نام‌های دو پارامتر آن، با نام‌های بکارگرفته شده در Html.TextBoxFor و Html.PasswordFor یکی هستند، با مقادیر ارسالی آن‌ها مقدار دهی شده و سپس در متد قابل استفاده خواهند بود. در پشت صحنه هم از همان رکوردهای موجود در Request.Form (یا سایر موارد ذکر شده)، استفاده می‌شود. در اینجا هر رکورد مثلا مجموعه Request.Form، کلیدی مساوی نام ارسالی به سرور را داشته و مقدار آن هم، مقداری است که کاربر وارد کرده است.

اگر همانندی یافت نشد، آن پارامتر با نال مقدار دهی می‌گردد. بنابراین اگر برای مثال یک پارامتر از نوع int را معرفی کرده باشید و چون نوع int، نال نمی‌پذیرد، یک استثناء بروز خواهد کرد. برای حل این مشکل هم می‌توان از Nullable types استفاده نمود (مثلا بجای int id نوشت int? id تا مشکلی جهت انتساب مقدار نال وجود نداشته باشد).

همچنین باید دقت داشت که این بررسی تطابق‌های بین نام عناصر HTML و نام پارامترهای متدها، case insensitive است و به کوچکی و بزرگی حروف حساس نیست. برای مثال، پارامتر معرفی شده در متد LoginResult مساوی string name است، اما نام

خاصیت تعریف شده در کلاس Account مساوی Name بود.

ج) استفاده از ویژگی جدیدی به نام Data Model Binding

در ASP.NET MVC چون می‌توان با یک Strongly typed view کار کرد، خود فریم ورک این قابلیت را دارد که اطلاعات ارسالی یکی فرم را به صورت خودکار به یک وهله از یک شیء نگاشت کند. در اینجا model binder وارد عمل می‌شود، مقادیر ارسالی را استخراج کرده (اطلاعات دریافتی از Form یا کوئری استرینگ‌ها یا اطلاعات مسیریابی و غیره) و به خاصیت‌های یک شیء نگاشت می‌کند. بدیهی است در اینجا این خواص باید عمومی باشند و هم نام عناصر HTML ارسالی به سرور. همچنین model binder پیش فرض ASP.NET MVC را نیز می‌توان کاملا تعویض کرد و محدود به استفاده از model binder توکار آن نیستیم.
وجود این Model binder، کار با ORMها را بسیار لذت بخش می‌کند؛ از آنجائیکه خود فریم ورک ASP.NET MVC می‌تواند عناصر شیءایی را که قرار است به بانک اطلاعاتی اضافه شود، یا در آن به روز شود، به صورت خودکار ایجاد کرده یا به روز رسانی نماید.
نحوه کار با model binder را در متد Login کنترلر فوق می‌توانید مشاهده کنید. بر روی return View آن یک breakpoint قرار دهید. فرم سوم را به سرور ارسال کنید و سپس در VS.NET خواص شیء ساخته شده را در حین دیباگ برنامه، بررسی نمائید.
بنابراین تفاوتی نمی‌کند که از چندین پارامتر استفاده کنید یا اینکه کلا یک شیء را به عنوان پارامتر معرفی نمائید. فریم ورک سعی می‌کند اندکی هوش به خرج داده و مقادیر ارسالی به سرور را به پارامترهای تعریفی، حتی به خواص اشیاء این پارامترهای تعریف شده، نگاشت کند.

در ASP.NET MVC سه نوع Model binder وجود دارند:
1) Model binder پیش فرض که توضیحات آن به همراه مثالی ارائه شد.
2) Form collection model binder که در ادامه توضیحات آن‌را مشاهده خواهید نمود.
3) HTTP posted file base model binder که توضیحات آن به قسمت بعدی موکول می‌شود.

یک نکته:
اولین متد LoginResult کنترلر را به نحو زیر نیز می‌توان بازنویسی کرد:

```
[HttpPost]
[ActionName("LoginResultWithFormCollection")]
public ActionResult LoginResult(FormCollection collection)
{
	string name = collection["name"];
	string password = collection["password"];

	if (name == "Vahid" && password == "123")
		ViewBag.Message = "Succeeded";
	else
		ViewBag.Message = "Failed";

	return View("Result");
}
```

در اینجا FormCollection به صورت خودکار بر اساس مقادیر ارسالی به سرور توسط فریم ورک تشکیل می‌شود (FormCollection هم یک نوع model binder ساده است) و اساسا یک NameValueCollection می‌باشد.
بدیهی است در این حالت باید نگاشت مقادیر دریافتی، به متغیرهای متناظر با آن‌ها، دستی انجام شود (مانند مثال فوق) یا اینکه می‌توان از متد UpdateModel کلاس Controller هم استفاده کرد:

```
[HttpPost]
public ActionResult LoginResultUpdateFormCollection(FormCollection collection)
{
	var account = new Account();
	this.UpdateModel(account, collection.ToValueProvider());

	if (account.Name == "Vahid" && account.Password == "123")
		ViewBag.Message = "Succeeded";
	else
```

```
                ViewBag.Message = "Failed";

        return View("Result");
}
```

متد توکار UpdateModel، به صورت خودکار اطلاعات FormCollection دریافتی را به شیء مورد نظر، نگاشت می‌کند.
همچنین باید عنوان کرد که متد UpdateModel، در پشت صحنه از اطلاعات Model binder پیش فرض و هر نوع Model binder سفارشی که ایجاد کنیم استفاده می‌کند. به این ترتیب زمانیکه از این متد استفاده می‌کنیم، اصلا نیازی به استفاده از FormCollection نیست و متد بدون آرگومان زیر هم به خوبی کار خواهد کرد:

```
[HttpPost]
public ActionResult LoginResultUpdateModel()
{
        var account = new Account();
         this.UpdateModel(account);

         if (account.Name == "Vahid" && account.Password == "123")
                ViewBag.Message = "Succeeded";
         else
                ViewBag.Message = "Failed";

         return View("Result");
}
```

استفاده از model binderها همینجا به پایان نمی‌رسد. نکات تکمیلی آن‌ها در قسمت بعدی بررسی خواهند شد.

نظرات خوانندگان

نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۱/۱۷ ۱۱:۴۸:۱۰

ممنون.
-1تو بعضی مثال ها بجای استفاده از [ActionName] از FormCollection بعنوان یکی از پارامترهای اکشن متد استفاده میشه(اگه اشتباه نکنم Scaffold هم از این روش استفاده میکنه).برای داشتن متد هم نام کدوم روش پیشنهاد میشه؟

-2همونطور که گفتید هنگام استفاده از ORM کار با ModelBinder لذت بخش میشه.با توجه به اینکه بیشتر برنامه های وب هم به این شکل هستن.ممنون میشم از روش های اول و دوم هم مثالی از دنیای واقعی بزنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۷ ۱۲:۱۹:۳۳

- Steven Sanderson (عضو تیم ASP.NET MVC) از روش ActionName استفاده می‌کنه (نویسنده کتاب Pro ASP.NET MVC هم هست). بنابراین روش برگزیده رو میشه این حالت درنظر گرفت. در مورد FormCollection هم در انتهای بحث توضیح دادم. Scaffold هم جهت تولید View مورد استفاده قرار می‌گیره نه کنترلر و اکشن متدهای آن. البته در این حالت strongly typed ، نیاز به مدل خواهد داشت که خارج از بحث FormCollection است؛ چون loosely typed است و نمیشه Viewایی رو از یک FormCollection استخراج کرد.
- هدف من تکمیل بحث بود. آشنایی با انواع روش‌های ممکن.
ضمن اینکه من چند سال قبل به کمک روش Request.Form که توضیح دادم، یک form generator برای ASP.NET Web forms نوشتم. زمانیکه کنترل‌های وب فرم‌ها به صورت پویا به صفحه اضافه بشن، دیگه eventها جهت دریافت مقادیر اون‌ها معنا نخواهند داشت (با توجه به اینکه کل صفحه رو بخواهیم پویا تولید کنیم و برنامه چند صد فرم پویا داشته باشد). در اینجا از همین روش Request.Form برای دریافت مقادیر کنترل‌های پویای اضافه شده به صفحه استفاده کردم.

نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۱/۱۷ ۱۵:۱۷:۲۹

با تشکر از شما.
الان که چک کردم Scaffold هم از [ActionName] استفاده میکنه.
ضمنا منظور من از Scaffold; قسمت اضافه کردن کنترلر(که در صورت استفاده از EF برا ما اکشن متدها و View های CURD رو میسازه) بود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۷ ۱۶:۰۳:۲۴

البته این Viewها (ی create/update و غیره) توسط scaffold template صفحه اضافه کردن view هم از روی یک مدل قابل تولید هستند (در لیست قالب‌ها هست). یعنی الزاما نیازی به EF نیست. چون EF مدل‌هاش دردسترس است این قابلیت برای اون هم وجود داره یا هر ORM دیگری.

نویسنده: Shima Besharat
تاریخ: ۱۳۹۱/۰۱/۱۸ ۰۰:۲۵:۵۹

سلام خدمت مهندس عزیز
من منظورتون از «روش ActionName رو میشه مرجع در نظر گرفت» متوجه نشدم منظورتون از روش ActionName کدوم روش است که در مطالب بالا توضیح داده اید؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۸ ۰۱:۱۶:۵۶

عنوان شد که اگر از FormCollection به عنوان پارامتر استفاده بشه، می‌شود دو متد هم نام را در یک کنترلر داشت. پاسخ این است که خیر. تحت هر حالتی نمی‌تونید دو متد هم نام را در یک کنترلر داشته باشید، مگر اینکه یکی httppost باشه و یکی httpget و امثال این حالت. بررسی آن هم ساده است. دو متد هم نام درست کنید. یکی با پارامتر و یکی بدون پارامتر. هر دو هم httppost. بعد سعی کنید یک فرم را به سرور ارسال کنید. با پیغام controller type is ambiguous مواجه خواهید شد. روش حل این مساله استفاده از ActionName است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۸ ۰۱:۳۸:۲۵

جهت تکمیل: می‌توان دو یا چند متد همنام در یک کنترلر داشت اگر از Verbهای مختلفی استفاده کنند. Post و Get و امثال این، Http Verb هستند.

نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۱/۱۸ ۱۴:۵۶:۰۱

کد کامپایل نمیشه +

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۸ ۱۶:۲۶:۰۲

بله. ویژگی‌های HttpPost و HttpGet مقدم نیستند بر اصول زبان مورد استفاده. اصول زبان مورد استفاده حین تعریف امضای متدها باید رعایت شوند (هر overloadیی رو نمیشه تعریف کرد). اما ... اگر ... این اصول رعایت شوند و کد شما کامپایل شود، آنگاه می‌توان دو یا چند overload را با verbهای مختلفی بدون مشکل دریافت خطای ambiguous استفاده کرد.

نویسنده: Mohsen
تاریخ: ۱۳۹۱/۰۱/۱۹ ۱۴:۴۷:۰۱

پس میشه گفت در واقع فریم ورک هر وقت با پارامتر (از نوع شی یا هر چیزی) روبرو می شود هم UpdateModel را فراخوانی می کند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۹ ۱۶:۱۴:۲۶

سورس MVC3 در دسترس است: (^)
روشی که در متد TryUpdateModel بکارگرفته شده (در فایل Controller.cs) که مبتنی بر کار با اینترفیس IModelBinder است، در قسمت‌های دیگر هم بکار رفته. بنابراین مکانیزم کلی یکی است. نام متدها شاید فرق کند.

نویسنده: امیر خلیلی
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۶:۱۶

در هنگام ارسال اطلاعات فرم همه پارامترها در url نمایش داده میشه
چطور میشه از نمایش این پارامترها جلوگیری کرد ؟
و در url تنها نام همان اکشن نمایش داده شود
و اینکه من از FormCollection برای دریافت مقادیر فرم استفاده می‌کنم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۶:۴۲

این مساله فقط به یک مورد مرتبط است و آن هم متد ارسال اطلاعات فرم است:

```
<form action="/Login/LoginResult" method="post">
```

اگر method فرم مساوی Get باشد ، پارامترهای ارسالی در Url ظاهر می‌شوند؛ در حالت **post** خیر:

```
@using(Html.BeginForm("action","ctrl", FormMethod.Post))
{
}
```

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۳/۰۱/۰۱ ۱:۵۹

وقتی method فرم رو روی Get تنظیم میکنم مقدار متغییرها در Controller دریافت نمیشه و به جای اون null دریافت میشه. کدی که استفاده کردم به شکل زیره :

```
@using (Html.BeginForm("LoginResultWithParams", "Login",FormMethod.Get))
{
    <fieldset>
        <legend>Test LoginResult(string name, string password)</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)
        </p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)
        </p>
        <input type="submit" value="Login" />
    </fieldset>
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۱ ۹:۲۴

برای فرم لاگین هیچ وقت از حالت Get استفاده نکنید. حالت Get نمونه استفاده‌اش در سایت جاری، در صفحه‌ی اول سایت، در تکست باکس جستجو است. عبارتی که کاربر وارد کرده، در کوئری استرینگ صفحه‌ی نتایج هم نمایش داده می‌شود. مزیت آن امکان به خاطر سپاری این Url و عبارت وارد شده و در آینده، استفاده مجدد از آن است.
آیا در مورد فرم لاگین نیز باید چنین کاری انجام شود و باید بتوان Url آن‌را به همراه Id و کلمه عبور کاربر، برای استفاده بعدی ذخیره کرد؟ خیر؛ به دلایل امنیتی این‌کار صحیح نیست.
در کل در این حالت خاص، به Url نهایی دقت کنید. نام کوئری استرینگ‌های آن باید با پارامترهای اکشن متد متناظر نهایی، تطابق داشته باشند. همچنین اگر بر روی اکشن متد آن، ویژگی HttpPost قرار گرفته باید حذف شود.