ASP.NET MVC Fundamentals

By: Mosh Hamedani

Exercise Hints

Creating Links

There are a few different ways to create links in ASP.NET MVC apps. The simplest way is to use raw HTML:

```
<a href="/Movies/Index">View Movies</a>
```

Alternatively, you can use the **ActionLink** method of **HtmlHelper** class:

```
@Html.ActionLink("View Movies", "Index", "Movies")
```

If the target action requires a parameter, you can use an anonymous object to pass parameter values:

```
@Html.ActionLink("View Movies", "Index", "Movies", new { id = 1 })
```

This should generate a link like the following:

/movies/index/1

But for a reason only known to programmers at Microsoft, this method doesn't generate this link, unless you pass another argument to the ActionLink. This argument can be null or an anonymous object to render any additional HTML attributes:

```
@Html.ActionLink("View Movies", "Index", "Movies", new { id = 1 }, null)
```

So, ActionLink or raw HTML, which approach is better? ActionLink queries the routing engine for the URL associated with the given action. If you have a custom URL associated with an action, and change that URL in the future, ActionLink will pick the latest URL, so you don't need to make any changes. But with raw HTML, you need to update your links when URLs are changed.

Having said, changing URLs is something you should avoid because these URLs are the public contract of your app and can be referenced by other apps or bookmarked by users. If you change them, all these bookmarks and external references will be broken.

So, at the end, whether you prefer to use raw HTML or @Html.ActionLink() is your personal choice.

HTML Tables

In the demo I showed you, I used a few CSS classes on my HTML table to give it the look and feel you saw in the video. These classes are part of Bootstrap, a front-end framework for building modern, responsive applications.

```
<table class="table table-bordered table-hover">
```

Type	**Helper Method**
ViewResult	**View()**

Type	Helper Method
PartialViewResult	**PartialView()**
ContentResult	**Content()**
RedirectResult	**Redirect()**
RedirectToRouteResult	**RedirectToAction()**
JsonResult	**Json()**
FileResult	**File()**
HttpNotFoundResult	**HttpNotFound()**
EmptyResult	

Action Parameters

Sources

- Embedded in the URL: /movies/edit/1
- In the query string: /movies/edit?id=1
- In the form data

Convention-based Routes

```
routes.MapRoute(
    “MoviesByReleaseDate”,
    “movies/released/{year}/{month}”,
    new {
         controller = “Movies”,
         action = “MoviesReleaseByDate”
    },
    new {
         year = @“\d{4}, month = @“\d{2}”
    }    isFavorite = false;
}
```

Attribute Routes

```
[Route(“movies/released/{year}/{month}”)
public ActionResult MoviesByReleaseDate(int year, int month)
{
}
```

To apply a constraint use a colon:

```
month:regex(\\d{2}):range(1, 12)
```

Passing Data to Views

Avoid using ViewData and ViewBag because they are fragile. Plus, you have to do extra casting, which makes your code ugly. Pass a model (or a view model) directly to a view:

```
return View(movie);
```

Razor Views

```
@if (…)
{
    // C# code or HTML
}
```

```
@foreach (…)
{
}
```

Render a class (or any attributes) conditionally:

```
@{
   var className = Model.Customers.Count > 5 ? “popular” : null;
}
<h2 class=“@className”>…</h2>
```

Partial Views

To render:

```
@Html.Partial(“_NavBar”)
```