educación IT
CODE YOUR FUTURE

Programación Web .NET Core

Módulo 7

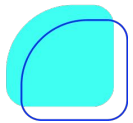
Métodos, enlazadores y controladores

Enlazador de modelos predeterminado

El *enlazador* asegura que los datos correctos sean enviados a los parámetros en un método de acción del controlador. Esto permite que ASP.NET Core MVC cree instancias del modelo que satisfacen la solicitud del usuario.

Por ejemplo, el enlazador de modelos predeterminado examina la definición de los parámetros de acción del controlador y los parámetros de solicitud para determinar los valores de solicitud para pasar a un parámetro de acción específico.

Este proceso de enlace del modelo puede ahorrar mucho tiempo a los desarrolladores y evitar varios errores inesperados en tiempo de ejecución que pueden surgir debido a parámetros incorrectos. ASP.NET Core MVC incluye un enlazador de modelos predeterminado con lógica sofisticada, que pasa los parámetros de manera correcta en casi todos los casos, sin un código personalizado complejo.



Para comprender el proceso de enlace del modelo predeterminado, considera la siguiente secuencia de pasos:

1. Un navegador web recibe la solicitud: ***http://www.contoso.com/product/display/45***.
2. Esta solicitud identifica tres aspectos:
    - La clase de modelo que le interesa al usuario. El usuario ha solicitado un producto.
    - La operación se hace sobre la clase modelo.
    - La instancia específica de la clase modelo. El usuario ha solicitado el producto con ID 45.
3. El tiempo de ejecución de ASP.NET Core MVC recibe la solicitud.
4. El tiempo de ejecución de ASP.NET Core MVC llama a una acción y le pasa los parámetros correctos.

En el ejemplo, el tiempo de ejecución de ASP.NET Core MVC llama a la acción **`Display`** en el controlador del **`Product`** y pasa el **`ID`** ***45*** como parámetro a la acción, para que se pueda mostrar el producto correcto. Para hacer esto, **el tiempo de ejecución de ASP.NET Core MVC usa enlazadores de modelos para determinar cómo se pasan los parámetros a su método de acción**.

Manejo de valores del formulario

Puedes sobrecargar un método de acción: crear dos métodos de acción con el mismo nombre que se diferenciarán por los parámetros que reciben. Un escenario común es una acción que carga un formulario y una acción que guarda los datos del `form`, ambas tienen el mismo nombre. El tiempo de ejecución de ASP.NET Core MVC, sabe qué método debe ser invocado por medio de los atributos siguientes:

- **`HttpPostAttribute`**
- **`HttpGetAttribute`**.

Puedes anotar una acción con el atributo llamado **`HttpGetAttribute`**, el cual indica que es una operación **`get`** que se suele usar para **cargar un formulario**.

También puedes anotar una acción con el atributo **`HttpPostAttribute`**, que indica que es una operación **`post`** que generalmente se usa para **guardar los datos de un formulario**.

El código a continuación muestra una clase de modelo simple:

Una clase de modelo

```
namespace ModelNamespace
{
public class Person
   {
public string FirstName{ get; set; }
public string LastName{ get; set; }
    }
}
```

El código de la siguiente pantalla muestra una clase controlador que contiene dos métodos de acción, ambos con la **nombreGetName**. Uno de ellos está anotado con el atributo **HttpGetAttribute**, y el otro con el atributo **HttpPostAttribute**.

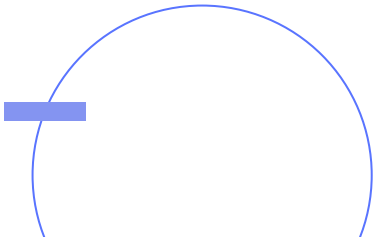
Uso de HttpGetAttribute y HttpPostAttribute

```
public class PersonController :Controller
{
   [Route("Person/GetName")]
[HttpGet]
public IActionResultGetName()
   {
return View();
   }
   [Route("Person/GetName")]
[HttpPost]
public IActionResultGetName(Personperson)
   {
return View("ShowName", person);
   }
}
```

El siguiente código muestra la vista a la que llama la acción **`GetName()`**:

El archivo *GetName.cshtml*

```
@model ModelNamespace.Person
@using (Html.BeginForm())
{
@Html.EditorForModel()
<input type="submit" value="Submit my name" />
}
```



El siguiente código muestra la vista a la que llama la acción **`GetName(Personperson)`**:

El archivo *ShowName.cshtml*

```
@model ModelNamespace.Person
Hello@Model.FirstName @Model.LastName
```

Si un usuario solicita la URL relativa siguiente: ***/Person/GetName***, se llama a la acción cuyo nombre es **`GetName()`**, que llama a la vista **`GetName`**. La vista `GetName` envía al navegador un **formulario** que contiene **dos cuadros de texto** y un **botón de envío**. Cuando un usuario completa las cajas de texto y hace clic en el botón ***Enviar***, ASP.NET en tiempo de ejecución de Core MVC, usa el enlazador de modelos para crear una instancia de tipo **`Person`** y llena sus propiedades en base a lo que el usuario ingresó en el formulario. Luego, en tiempo de ejecución de ASP.NET Core MVC, se llama al **`GetName (Personperson)`**, pasándole la instancia de tipo **`Person`**.

La acción **`GetName(persona persona)`** llama a la vista **`ShowName`**. La vista **`ShowName`** envía los valores de las propiedades de la *Persona* modelo al navegador.

Agregar operaciones CRUD a los controladores

A menudo, un controlador tiene operaciones **CRUD**. Las operaciones de **creación** se usan para crear o agregar nuevos artículos. Las operaciones de **lectura** se utilizan para leer, recuperar, buscar o ver entradas existentes. Las operaciones de **actualización** se utilizan para editar entradas existentes. Las operaciones de **eliminación** se usan para eliminar entradas existentes.

El ejemplo de las pantallas siguientes, muestra un controlador con ***Operaciones CRUD***.

```
public class PersonController :Controller
{
   [HttpGet]
public IActionResultIndex()
   {
   // TODO: Addlogichere
return View(people);
   }
   [HttpGet]
public IActionResultDetails(int id)
   {
    // TODO: Addlogichere
return View(person);
   }
   [HttpGet]
public IActionResultCreate()
   {
   // TODO: Addlogichere
returnView();
   }
```

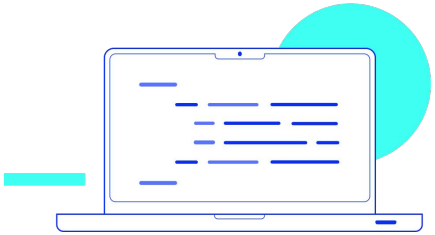
...

```
...
   [HttpPost]
public IActionResultCreate(Personperson)
   {
   // TODO: Addlogichere
return RedirectToAction("Index");
   }
   [HttpGet]
public IActionResultEdit(int id)
   {
   // TODO: Addlogichere
return View(person);
   }
   [HttpPost]
public IActionResultEdit(int id, Personperson)
   {
   // TODO: Addlogichere
return RedirectToAction("Index");
   }
...
```

```
   [HttpGet]
public IActionResultDelete(int id)
   {
   // TODO: Addlogichere
return View(person);
   }
   [HttpPost, ActionName("Delete")]
public IActionResultDeleteConfirmed(int id)
  {
   // TODO: Addlogichere
return RedirectToAction("Index");
   }
}
```

En el ejemplo anterior, hay dos operaciones de **lectura**: el método **`Index`** devuelve una colección de personas, mientras que el método **`Details`** devuelve una persona específica. Hay dos operaciones de **creación**: **`Create()`**. El método permite al usuario ingresar las propiedades de la persona en **`Create`** y **`Create(Personperson)`** y crea una persona basada en las propiedades ingresadas por el usuario. Hay dos operaciones de **actualización**: el método **`Edit(int id)`** permite al usuario cambiar las propiedades de una persona existente, mientras que el **`Edit(int id, Personperson)`** guarda los cambios realizados por el usuario en la persona.

Hay dos **eliminar operaciones** que aceptan el mismo `id` de parámetro. Dado que las dos operaciones de eliminación tienen el mismo nombre y parámetro, y es ilegal tener dos métodos en C# con la misma firma, el nombre del segundo es la acción **`DeleteConfirmed`**.



educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!