educación IT
CODE YOUR FUTURE

Programación Web .NET Core

Módulo 5

Uso de rutas y modelos

Uso de rutas para pasar parámetros

Puede acceder a los valores de las variables mediante dos métodos:

- Con la colección **RouteData.Values**.

```
    public IActionResultPrint()
    {
string id = (string)RouteData.Values["id"];
        return Content("id: " + id);
    }
```

- Utilizando el **enlace de modelos** para pasar los parámetros a las acciones.

```
    public IActionResultPrint(string id)
    {
        return Content("id: " + id);
    }
```

El motor de enrutamiento separa la URL relativa en una solicitud, en uno o más segmentos.

Si deseas que **uno de los segmentos especifique el nombre del controlador,** puedes utilizar la variable de segmento **`{controller}`**. Esta variable es interpretada como el controlador que se va a instanciar.

Para **fijar una ruta a un solo controlador,** puedes especificar un valor para la variable en la propiedad `defaults`. De una manera similar, si deseas que **uno de los segmentos especifique el método de acción,** puedes utilizar la variable de segmento de **`{acción}`**. El invocador de la acción siempre interpreta esta variable como la acción a llamar.

Para **fijar una ruta a una sola acción,** se puede especificar un valor para la variable de acción en la propiedad por defecto.

Las variables de segmento o los valores predeterminados con otros nombres no tienen un significado especial para **MVC**.

Es posible acceder a los valores de estas variables mediante uno de dos métodos: el **`RouteData.Values`** o el **enlace de modelo** para pasar valores a los parámetros de acción.

Uso de la colección RouteData.Values

En el método de acción, se puede acceder a los valores de cualquier variable de segmento utilizando **`RouteData.Values`** colección.

Por ejemplo, se puede acceder al nombre de la clase del controlador utilizando la declaración **`RouteData.Values["controller"]`**, y al nombre del método de acción con la declaración de **`RouteData.Values["action"]`**.

El código de la derecha muestra una acción que usa la colección **`RouteData.Values`**:

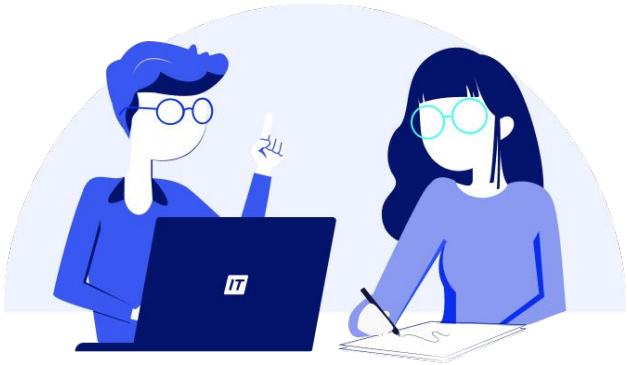
```
public class ExampleController : Controller
{
   public IActionResultPrint()
   {
     string controller =
(string)RouteData.Values["Controller"];
     string action =
(string)RouteData.Values["Action"];
     return Content("Controller: " +
controller + ". Action: " + action);
   }
}
```

El siguiente código muestra una ruta.

Ejemplo de ruta

```
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "someRoute",
   template: "{controller}/{action}/{id?}",
   defaults: new { controller = "Example",
action = "Print" });
});
```

Si un usuario solicita la URL relativa **/Example/Print**, el controlador es: *Example*. En la acción: la cadena ***Print*** es lo que se mostrará en el navegador.



De manera similar, se puede usar la colección **RouteData.Values** para acceder a otras variables de segmento.

El siguiente código muestra un controlador que usa la colección **RouteData.Values** para acceder a la identificación valor del segmento:

Uso de la colección RouteData.Values

```
public class ExampleController : Controller
{
  public IActionResultPrint()
  {
string id = (string)RouteData.Values["id"];
return Content("id: " + id);
  }
}
```

Si un usuario solicita la **URL** relativa **Example/Print/1**, la siguiente cadena aparece en la pantalla: **“id: 1”**.

Uso de modelos para obtener parámetros

El invocador de acciones MVC predeterminado pasa los parámetros apropiados a las acciones. Para ello examina la definición de la acción a la que debe pasar parámetros. El mecanismo de vinculación del modelo busca valores en la solicitud, que se pueden pasar como parámetros por nombre. Una de las ubicaciones que busca es la colección **`RouteData.Values`**.

Por lo tanto, puede utilizar el enlazador de modelos predeterminado para pasar parámetros a la acción. Si el nombre de un parámetro de método de acción coincide con el de una variable de segmento de ruta, el enrutador del modelo pasa el parámetro automáticamente.

El siguiente código muestra un controlador que usa el enlace de modelo para acceder al valor del segmento de identificación:

Usar enlace de modelo

```
public class ExampleController : Controller
{
   public IActionResultPrint(string id)
   {
      return Content("id: " + id);
    }
}
```

Si un usuario solicita la **URL** relativa **Example/Print/1**, la acción devuelve la siguiente cadena: **`id: 1`**.

Los parámetros del método de acción no tienen que ser del tipo de cadena. Por ejemplo, se puede cambiar la acción `Print` para obtener un `int` como parámetro.

El siguiente código muestra una acción que obtiene un `int` como parámetro:

Un ejemplo de enlace de modelo

```
public class ExampleController : Controller
{
   public IActionResultPrint(int id)
  {
      return Content("id: " + id);
  }
}
```

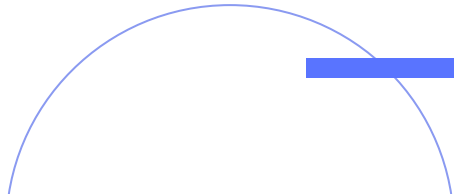
Si un usuario solicita la URL relativa **/Example/Print/1**, se devolverá la cadena **`id: 1`**. Si solicita URL relativa **Example/Print/1**, se devolverá la cadena **`id: 0`**. Esto se debe a que **`int` es un tipo de valor, que no puede almacenar valores nulos.**

Puedes permitir que los tipos de valor tengan valores nulos si los marcas como **`Nullable`**. Por ejemplo, si cambias la acción **`Print`** para obtener un parámetro de tipo **`int?`** y luego un usuario solicita la URL relativa **/Example/Print/1**, el valor nulo se asignará a **`id`**.

El siguiente código muestra una acción que obtiene un parámetro de tipo **`int?`**:

Un parámetro que acepta valores NULL

```
public class ExampleController : Controller
{
    public IActionResultPrint(int?id)
    {
        return Content("id: " + id);
    }
}
```



También puedes permitir que un parámetro de una acción tenga valores opcionales.

El siguiente código muestra una acción que obtiene un parámetro con un valor opcional:

El parámetro opcional

```
public class ExampleController : Controller
{
public IActionResultPrint(int id = 444)
{
return Content("id: " + id);
}
}
```

Si un usuario solicita las URLs relativas **/Example/Print/a** y **/Example/Print**, la cadena **`id: 444`** será devuelta.

Si un usuario solicita la URL relativa **/Example/Print/1**, se devolverá la cadena **`id: 1`**.

Puedes establecer varios segmentos en la plantilla de una ruta, y todos ellos pueden coincidir con los parámetros de una acción.

El siguiente código muestra una plantilla con varios segmentos:

Ruta con varios tramos

```
app.UseMvc(routes =>
{
routes.MapRoute(
        name: "someRoute",
        template:
"{controller}/{action}/{id}/{title?}",
        defaults: new { controller = "Example",
action = "Print" });
});
```

Al usar la ruta anterior, puedes escribir una acción que obtenga dos parámetros, **`id`** y **`title`**. Ambos parámetros pueden coincidir con los segmentos en la solicitud **URL**.

El siguiente código muestra una acción que tiene dos parámetros:

```
public class ExampleController : Controller
{

    public IActionResultPrint(string id,
string title)
    {
        return Content("id: " + id + ". title:
" + title);
    }
}
```

Si un usuario solicita la **URL** relativa **/Example/Print** la solicitud fallará porque el segmento de **`id`** en la ruta es obligatorio.

Si un usuario solicita la **URL** relativa **/Example/Print/1**, la solicitud tendrá éxito porque el segmento del **`title`** en la ruta es opcional.

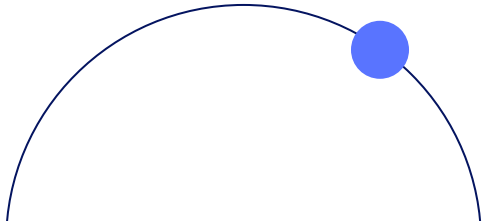
En este caso, el parámetro **`id`** tendrá el valor 1 y el parámetro de **`title`** tendrá el valor nulo.

Si un usuario solicita la URL relativa **/Example/Print/SomeTitle**, la solicitud se realizará correctamente. En este caso, el parámetro **`id`** tendrá el valor 1 y el parámetro title tendrá el valor **`SomeTitle`**.

Configurar rutas mediante atributos

Previamente, mencionamos que **ASP.NET Core MVC** admite dos tipos de enrutamiento, **enrutamiento basado en convenciones** y basado en **atributos de enrutamiento**.

Como vimos anteriormente, el enrutamiento basado en convenciones se configura en un lugar centralizado en la clase ***Startup.cs***.



El enrutamiento basado en atributos, como su nombre lo indica, permite configurar rutas usando atributos. **La principal ventaja de este tipo de enrutamiento es que los atributos permiten definir las rutas en el mismo controlador.**

```
[Route("Some")]
    public IActionResultSomeMethod()
    {
        return Content("Some method");
    }
```

> **Recuerda:** El enrutamiento basado en convenciones y el enrutamiento basado en atributos se pueden utilizar en la misma aplicación.

El siguiente código muestra la clase **Startup**, configurada para usar **MVC** sin rutas centralizadas:

```
public class Startup
{
   public void
ConfigureServices(IServiceCollectionservices)
   {
services.AddMvc();
   }
   public void Configure(IApplicationBuilderapp)
  {
app.UseMvc();
  }
}
```

Se puede utilizar el atributo de ruteo para especificar la ruta en una acción específica.

Por ejemplo: un controlador llamado **MyController** contiene una acción llamada **SomeMethod**.

Si un usuario solicita la URL relativa **/Some**, la solicitud fallará. Sin embargo, si se agrega **[Route("Some")]** encima de la declaración de la acción **SomeMethod**, la solicitud se realizará correctamente. El siguiente código muestra cómo configurar una ruta mediante un atributo:

Enrutamiento basado en atributos

```
public class MyController : Controller
{
[Route("Some")]
    public IActionResultSomeMethod()
 {
          return Content("Some method");
     }
}
```

Pasar parámetros mediante el enrutamiento de atributos

El enrutamiento de atributos es útil para pasar parámetros a una acción.

Por ejemplo, si la acción **SomeMethod** debe obtener un parámetro llamado **param**, se puede establecer un atributo de la siguiente manera: **[Route ("Mi/{param}")]**.

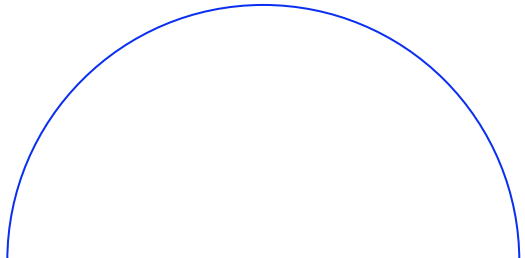
Después de configurar el atributo, si un usuario solicita la URL relativa **/Mi/val**, la solicitud tendrá éxito y `param` tendrá el valor `val`.

El siguiente código muestra cómo configurar una ruta de atributo con un parámetro. Se debe tener en cuenta que los componentes de la ruta tienen un nombre diferente al del método de acción:

Una ruta de atributo con parámetro

```
public class MyController : Controller
{
[Route("Mi/{param}")]
    public IActionResultSomeMethod(string param)
    {
            return Content(param);
    }
}
```

El atributo `Route` se puede utilizar para obtener varios segmentos de variables. El atributo de ruta (**`route`**) también se puede utilizar para asegurarse de que un segmento de una variable sea de un tipo específico. Por ejemplo, establecer el atributo **`[Route("Mi/{param1}/{param2:int}"]`** configura una ruta en la que deben existir dos segmentos de variables, **`param1`** y **`param2`**. **`“param2”`** debe ser de tipo **`int`**.



El siguiente código muestra cómo configurar un atributo de ruteo con dos parámetros, donde el segundo debe ser de tipo **int**:

Una ruta de atributo con dos parámetros

```
public class MyController : Controller
{
[Route("Mi/{param1}/{param2:int}")]
    public IActionResultSomeMethod(string param1,
int param2)
    {
          return Content("param1: " + param1 + ",
param2: " + param2);
     }
}
```

Una ruta de atributo con dos parámetros

```
public class MyController : Controller
{
[Route("Mi/{param1}/{param2?}")]
    public IActionResultSomeMethod(string param1,
string param2)
    {
          return Content("param1: " + param1 + ",
param2: " + param2);
    }
}
```

Según el ejemplo anterior, si un usuario solicita la **URL relativa Mi/test**, la solicitud se realizará correctamente porque **param2** es opcional. En este caso, el valor de **param2** será nulo. Por otro lado, si un usuario solicita la URL relativa **Mi**, la solicitud fallará porque **param1** no es opcional.

Decorar una clase controlador con un atributo de ruta

Normalmente, todas las rutas en una clase controlador comienzan con el mismo prefijo.

El siguiente código muestra dos rutas con el mismo prefijo:

Dos rutas con el mismo prefijo

```
public class MyController : Controller
{
[Route("Mi/Metodo1")]
   public IActionResult Method1()
   {
       return Content("Method1");
   }
[Route("Mi/Metodo2")]
   public IActionResult Method2()
   {
      return Content("Method2");
   }
}
```

Puedes establecer un prefijo común para toda una clase controlador utilizando el atributo `[Route]` sobre la clase de controlador.

El siguiente código muestra cómo se puede aplicar un atributo de ruta a una clase de controlador:

Un atributo de ruta en una clase controlador

```
[Route("Mi")]
public class MyController : Controller
{
[Route("Metodo1")]
   public IActionResult Method1()
   {
         return Content("Method1");
   }
[Route("Metodo2")]
   public IActionResult Method2()
   {
        return Content("Method2");
   }
}
```

Según el ejemplo anterior, si un usuario solicita la URL relativa **/Mi/Metodo1**, la solicitud se realizará correctamente.

Sin embargo, si un usuario solicita la URL relativa **/Metodo1**, la solicitud fallará.

El código de la derecha muestra una ruta basada en convenciones:

Ruta basada en convenciones

```
app.UseMvc(routes =>
{
routes.MapRoute(
name: "default",
template: "{controller}/{action}");
});
```

Considera una clase de controlador llamada **`MiController`** que tiene dos acciones, una acción llamada **`Metodo1`** que tiene el atributo de ruta **`[Route("SomeRoute")]`** y una acción denominada **`Metodo2`** sin atributo de Ruta.

El siguiente código muestra una clase de controlador con dos acciones, una con un atributo de ruta y la segunda sin un atributo de ruta:

La clase Controller

```
public class MyController : Controller
{
[Route("SomeRoute")]
   public IActionResult Metodo1()
   {
           return Content("Metodo1");
   }
   public IActionResult Metodo2()
   {
          return Content("Metodo2");
   }
}
```

Cuando un usuario solicita la URL relativa **/SomeRoute**, se ejecuta la acción **`Metodo1`**. Esto se debe al atributo de ruta que está configurado por encima del **`Metodo1`**. Cuando un usuario solicita la URL relativa **/Mi/Metodo2**, se ejecuta la acción **`Metodo2`**. Esto se debe a que la ruta está centralizada en la clase ***Startup.cs***.

Puede usarse el enrutamiento de atributos con atributos **`Http [Verb]`** en lugar de usar el atributo **`Route`**.

Los atributos **`Http [Verb]`** deben usarse cuando desees limitar el acceso a la acción a un verbo **HTTP** específico.

Por ejemplo, si deseas que una acción se ejecute sólo cuando el **verbo HTTP** es **`GET`**, puedes usar el atributo **`HttpGet`**. De manera similar, si deseas que una acción se ejecute sólo cuando el **verbo HTTP** es **`POST`**, puedes usar el Atributo **`HttpPost`**.

El siguiente código muestra cómo se puede aplicar un atributo **`HttpGet`** a un método de acción:

Usar el atributo HttpGet

```
public class CitiesController : Controller
{
[HttpGet("/cities/{id}")]
    public IActionResultGetCity(int id) { ...
}
}
```

Cuando un usuario solicita la URL relativa **/cities/1**, se ejecuta la **`acciónGetCity`**. Esta acción solo coincide con el verbo **HTTP `GET`** porque está decorado con el atributo **`HttpGet`**.

Para hacer que el enrutamiento de atributos sea menos repetitivo, se puede aplicar el atributo de ruta a un controlador y aplicar el atributo **`Http [Verb]`** a una acción. En tales casos, el atributo **`route`** se utiliza para establecer el prefijo común.

El siguiente código muestra cómo puede combinar el atributo **`Route`** con los atributos **`Http [Verb]`**:

Combina el atributo Route con los atributos Http [Verb]

```
[Route("cities")]
public class CitiesController : Controller
{
[HttpGet]
   public IActionResultListCities() { ... }
[HttpGet("{id}")]
   public ActionResultGetCity(int id) { ... }
}
```

Cuando un usuario solicita la URL relativa **/cities**, se ejecuta la acción **`ListCities`**. Cuando un usuario solicita la URL relativa **/cities/1**, se ejecuta la acción **`GetCity`**.

¡Sigamos trabajando!