
epam
ASP.NET MVC BASICS:
MODEL BINDING
2017

MODEL BINDING BASICS

- Предположим, что в контроллере определен следующий **action**:

```
public class HomeController : Controller
{
    private readonly IPersonRepository _personRepository;

    public ViewResult Person(int id)
    {
        Person myPerson = _personRepository.GetPersonById(id);
        return View(myPerson);
    }
}
```

MODEL BINDING BASICS

- Таблица маршрутов состоит только из маршрута по умолчанию, созданного автоматически. Для напоминания:

```
routes.MapRoute(
     name: "Default",
     url: "{controller}/{action}/{id}",
     defaults: new
     {
          controller = "Home",
          action = "Index",
          id = UrlParameter.Optional
     }
);
```


MODEL BINDING
Superdry
Superdry

MODEL BINDER

Основным механизмом **model binding** является интерфейс **IModelBinder**, который описывает механизм связывания данных:

```
namespace System.Web.Mvc
{
    public interface IModelBinder
    {
        object BindModel(
            ControllerContext controllerContext,
            ModelBindingContext bindingContext
        );
    }
}
```

VALUE PROVIDERS

При формировании запроса клиент может передавать данные серверу из разных источников, таких как:

- Form body
- Query string
- Route segments
- Files
- Cookies
- Request headers

ASP.Net MVC «из коробки» предоставляет следующие источники данных (**value providers**):

- `FormValueProvider` — Form body
- `QueryStringValueProvider` — Query string
- `RouteDataValueProvider` — Route segments
- `HttpFileCollectionValueProvider` — Files
- `CookieValueProvider` — Cookies
- и другие

PRIMITIVE TYPE BINDING

- Чтобы избежать проблем, параметры можно модифицировать. Можно использовать тип, допускающий значения **null**, как показано ниже:

```
public ViewResult RegisterPerson(int? id) { ... }
```

- При таком подходе значение параметра **id** будет равно **null**, если в запросе не обнаружено подходящих для преобразования данных. Или можно сделать необязательным, предоставив значение по умолчанию:

```
public ViewResult RegisterPerson(int id = 23) { ... }
```

COMPLEX TYPE BINDING

```
public class Person
{
    public int PersonId { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }

    public DateTime BirthDate { get; set; }

    public Address HomeAddress { get; set; }
}
```

COMPLEX TYPE BINDING

```
public class Address
{
    public string Line1 { get; set; }

    public string Line2 { get; set; }

    public string City { get; set; }

    public string PostalCode { get; set; }

    public string Country { get; set; }
}
```

COMPLEX TYPE BINDING

```
@Html.EditorFor(m => m.FirstName)

<input class="text-box single-line" id="FirstName" name="FirstName"
type="text" value="Joe" />

@Html.EditorFor(m => m.HomeAddress.Line1)

<input class="text-box single-line" id="HomeAddress_Line1"
name="HomeAddress.Line1" type="text" value="123 North Street" />
```

COMPLEX TYPE BINDING

```
@using MvcApp.Models;
@model MvcApp.Models.Person
@{
    Person myPerson = new Person() { FirstName = "Jane", LastName = "Doe“ };
}
@using (Html.BeginForm()) {
    @Html.EditorFor(m => m.FirstName)
    @Html.EditorFor(m => myPerson.FirstName)
    <input type="submit" value="Submit" />
}

<!-- будет представлено как -->
<input class="text-box single-line" id="FirstName" name="FirstName" type="text"
value="Joe" />
<input class="text-box single-line" id="myPerson_FirstName"
name="myPerson.FirstName" type="text" value="Jane" />
```

COMPLEX TYPE BINDING

Если метод действия, которому отправлена форма, имеет следующую сигнатуру:

```
public ActionResult Index(Person firstPerson, Person myPerson) { ... }
```

то объект первого параметра будет привязан с применением данных без префикса, а для объекта второго параметра будет производиться поиск данных, которые начинаются с имени параметра, т.е. **myPerson.FirstName**, **myPerson.LastName** и т.д.

COLLECTION BINDING

```
@{
    ViewBag.Title = "My favorite books";
}
Enter your four favorite books:
@using (Html.BeginForm())
{
    @Html.TextBox("books")
    @Html.TextBox("books")
    @Html.TextBox("books")
    <input type=submit />
}
```

COLLECTION BINDING

Здесь с использованием вспомогательного метода **@Html.TextBox()** создаются четыре элемента. Они имеют в атрибуте **name** значение **books**:

```
<input id="books" name="books" type="text" value="" />
<input id="books" name="books" type="text" value="" />
<input id="books" name="books" type="text" value="" />
<input id="books" name="books" type="text" value="" />
```

Получить значения, введенные пользователем, можно с помощью **action**, подобного следующему:

```
[HttpPost]
public ViewResult Books(List<string> books) { ... }
```

COMPLEX TYPE COLLECTION BINDING

Трюк с привязкой множественных значений очень хорош, но если мы хотим работать со специальными типами, то должны генерировать HTML-разметку в определенном формате:

```
@model List<MvcApp.Models.Person>
@for (int i = 0; i < Model.Count; i++)
{
    <h4>Person Number: @i</h4>
    @:First Name: @Html.EditorFor(m => m[i].FirstName)
    @:Last Name: @Html.EditorFor(m => m[i].LastName)
}
```

COMPLEX TYPE COLLECTION BINDING

Шаблонизированный вспомогательный метод генерирует HTML-разметку, в которой имя каждого свойства предваряется индексом соответствующего объекта в коллекции:

```
<h4>Person Number: 0</h4>
First Name: <input class="text-box single-line" name="[0].FirstName"
type="text" value="Joe" />
Last Name: <input class="text-box single-line" name="[0].LastName"
type="text" value="Smith" />

<h4>Person Number: 1</h4>
First Name: <input class="text-box single-line" name="[1].FirstName"
type="text" value="John" />
Last Name: <input class="text-box single-line" name="[1].LastName"
type="text" value="Doe" />
```

COMPLEX TYPE COLLECTION BINDING

Для привязки к данным понадобится определить метод действия, который принимает коллекцию необходимых типов как параметр:

```
[HttpPost]
public ViewResult Register (List<Person> people) { ... }
```

Поскольку осуществляется привязка к коллекции, **model binder** по умолчанию будет искать значения для свойств класса **Person**, имеющих префикс в виде индекса.

COMPLEX TYPE COLLECTION BINDING

Разумеется, использовать для генерации HTML-разметки шаблонизированные вспомогательные методы не обязательно.
Это можно делать явным образом во **view**:

```
<h4>First Person</h4>
First Name: @Html.TextBox("[0].FirstName")
Last Name: @Html.TextBox("[0].LastName")
<h4>Second Person</h4>
First Name: @Html.TextBox("[1].FirstName")
Last Name: @Html.TextBox("[1].LastName")
```

ANY QUESTIONS?

THANKS FOR YOUR ATTENTION!