## Передача данных в представления

Существует несколько подходов к передаче данных в представления:

* Строго типизированные данные: viewmodel
* Слабо типизированные данные
  + - ViewData (ViewDataAttribute)
    - ViewBag

### Строго типизированные данные (viewmodel)

Наиболее надежный подход — указание типа [модели](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/models/model-binding?view=aspnetcore-2.2) в представлении. Такая модель называется viewmodel. Экземпляр типа viewmodel передается в представление из действия.

Использование viewmodel для передачи данных в представление позволяет ему применять строгую проверку типов. Термин строгая типизация (или строго типизированный) означает, что каждая переменная и константа имеет явным образом определенный тип (например, string, int или DateTime). Допустимость типов, используемых в представлении, проверяется во время компиляции.

В [Visual Studio](https://www.visualstudio.com/vs/) и [Visual Studio Code](https://code.visualstudio.com/) строго типизированные члены классов перечисляются с помощью функции [IntelliSense](https://docs.microsoft.com/ru-ru/visualstudio/ide/using-intellisense). Чтобы просмотреть свойства viewmodel, введите имя переменной viewmodel с точкой (.) после него. Это позволяет писать код быстрее, допуская меньше ошибок.

Укажите модель с помощью директивы @model. Используйте модель с @Model:

**CSHTML**

@model WebApplication1.ViewModels.Address

<h2>Contact</h2>

<address>

@Model.Street<br>

@Model.City, @Model.State @Model.PostalCode<br>

<abbr title="Phone">P:</abbr> 425.555.0100

</address>

Чтобы предоставить модель для представления, контроллер передает ее в качестве параметра:

**C#**

public IActionResult Contact()

{

ViewData["Message"] = "Your contact page.";

var viewModel = new Address()

{

Name = "Microsoft",

Street = "One Microsoft Way",

City = "Redmond",

State = "WA",

PostalCode = "98052-6399"

};

return View(viewModel);

}

В отношении типов моделей, которые можно предоставлять для представления, ограничений нет. Мы рекомендуем использовать модели представлений POCO, которые позволяют определять минимум методов или не определять их вовсе. Как правило, классы viewmodel хранятся либо в папке Models, либо в отдельной папке ViewModels в корне приложения. Модель представления Address, используемая в приведенном выше примере, — это модель POCO, хранящаяся в файле с именем Address.cs:

**C#**

namespace WebApplication1.ViewModels

{

public class Address

{

public string Name { get; set; }

public string Street { get; set; }

public string City { get; set; }

public string State { get; set; }

public string PostalCode { get; set; }

}

}

Ничто не мешает вам использовать одни и те же классы как для типов viewmodel, так и для типов бизнес-модели. Однако применение отдельных моделей позволяет изменять представления независимо от компонентов бизнес-логики и доступа к данным в приложении. Разделение моделей и моделей представлений также предоставляет преимущества в плане безопасности, когда модели используют [привязку модели](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/models/model-binding?view=aspnetcore-2.2) и [проверку](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/models/validation?view=aspnetcore-2.2) для данных, отправляемых в приложение пользователем.

### Нестрого типизированные данные (ViewData, атрибут ViewData и ViewBag)

Помимо строго типизированных представлений, представления имеют доступ к нестрого типизированной (слабо типизированной) коллекции данных. В отличие от строгих типов, нестрогие типы (или слабые типы) предполагают, что тип используемых данных не объявляется явным образом. Коллекцию нестрого типизированных данных можно использовать для передачи небольших объемов данных в контроллеры и представления или из них.

| **Передача данных между...Passing data between a ...** | **ПримерExample** |
| --- | --- |
| Контроллером и представлением | Заполнение раскрывающегося списка данными. |
| Представлением и [представлением макета](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/views/layout?view=aspnetcore-2.2) | Задание содержимого элемента **<title>** в представлении макета из файла представления. |
| [Частичным представлением](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/views/partial?view=aspnetcore-2.2) и представлением | Мини-приложение, в котором данные выводятся на основе запрошенной пользователем веб-страницы. |

На эту коллекцию можно ссылаться в контроллерах и представлениях посредством свойства ViewData или ViewBag. Свойство ViewData представляет собой словарь нестрого типизированных объектов. Свойство ViewBag представляет собой оболочку для свойства ViewData, которая предоставляет динамические свойства для базовой коллекции ViewData.

Свойства ViewData и ViewBag разрешаются динамически во время выполнения. Так как они не обеспечивают проверку во время компиляции, их использование обычно приводит к большему числу ошибок, чем использование viewmodel. По этой причине некоторые разработчики предпочитают как можно реже использовать свойства ViewData и ViewBag или не использовать их вовсе. For that reason, some developers prefer to minimally or never use ViewData and ViewBag.

**ViewData**

ViewData — это объект [ViewDataDictionary](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.viewdatadictionary), доступ к которому осуществляется посредством ключей string. Строковые данные можно сохранять и использовать напрямую без приведения, однако при извлечении других значений объекта ViewData их необходимо приводить к соответствующим типам. С помощью свойства ViewData можно передавать данные, включая [частичные представления](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/views/partial?view=aspnetcore-2.2) и [макеты](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/views/layout?view=aspnetcore-2.2), из контроллеров в представления и внутри представлений.

Ниже приведен пример, в котором с помощью свойства ViewData в действии задаются значения для приветствия и адреса.

**C#**

public IActionResult SomeAction()

{

ViewData["Greeting"] = "Hello";

ViewData["Address"] = new Address()

{

Name = "Steve",

Street = "123 Main St",

City = "Hudson",

State = "OH",

PostalCode = "44236"

};

return View();

}

Работа с данными в представлении:

**CSHTML**

@{

// Since Address isn't a string, it requires a cast.

var address = ViewData["Address"] as Address;

}

@ViewData["Greeting"] World!

<address>

@address.Name<br>

@address.Street<br>

@address.City, @address.State @address.PostalCode

</address>

**Атрибут ViewData**

Другой подход, который использует [ViewDataDictionary](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.viewdatadictionary), — [ViewDataAttribute](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.mvc.viewdataattribute). Свойства на контроллерах или моделях страниц Razor, отмеченные атрибутом [ViewData], обладают своими собственными значениями, загружаемыми из словаря.

В следующем примере контроллер Home содержит свойство Title, отмеченное атрибутом [ViewData]. Метод About задает заголовок для представления About:

**C#**

public class HomeController : Controller

{

[ViewData]

public string Title { get; set; }

public IActionResult About()

{

Title = "About Us";

ViewData["Message"] = "Your application description page.";

return View();

}

}

В представлении About доступ к свойству Title осуществляется как доступ к свойству модели:

**CSHTML**

<h1>@Model.Title</h1>

В макете заголовок считывается из словаря ViewData.

CSHTML

<!DOCTYPE html>

<html lang="en">

<head>

<title>@ViewData["Title"] - WebApplication</title>

...

**ViewBag**

ViewBag — это объект [DynamicViewData](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.internal.dynamicviewdata), который обеспечивает динамический доступ к объектам, хранящимся в ViewData. Работать со свойством ViewBag может быть удобнее, так как оно не требует приведения. В приведенном ниже примере демонстрируется использование свойства ViewBag с тем же результатом, что и свойства ViewData ранее.

**C#**

public IActionResult SomeAction()

{

ViewBag.Greeting = "Hello";

ViewBag.Address = new Address()

{

Name = "Steve",

Street = "123 Main St",

City = "Hudson",

State = "OH",

PostalCode = "44236"

};

return View();

}

**CSHTML**

@ViewBag.Greeting World!

<address>

@ViewBag.Address.Name<br>

@ViewBag.Address.Street<br>

@ViewBag.Address.City, @ViewBag.Address.State @ViewBag.Address.PostalCode

</address>

**Одновременное использование ViewData и ViewBag**

Так как свойства ViewData и ViewBag ссылаются на одну и ту же базовую коллекцию ViewData, вы можете использовать ViewData и ViewBag вместе в различных сочетаниях при чтении и записи значений.

Например, задайте заголовок с помощью свойства ViewBag, а описание с помощью свойства ViewData в начале представления About.cshtml:

**CSHTML**

@{

Layout = "/Views/Shared/\_Layout.cshtml";

ViewBag.Title = "About Contoso";

ViewData["Description"] = "Let us tell you about Contoso's philosophy and mission.";

}

В этом примере свойства считываются, причем очередность использования свойств ViewData и ViewBag обратная. В файле \_Layout.cshtml получите заголовок с помощью свойства ViewData, а описание с помощью свойства ViewBag:

**CSHTML**

<!DOCTYPE html>

<html lang="en">

<head>

<title>@ViewData["Title"]</title>

<meta name="description" content="@ViewBag.Description">

...

Помните, что строки не требуют приведения значения ViewData. @ViewData["Title"] можно использовать без приведения.

Свойства ViewData и ViewBag можно использовать одновременно в различных сочетаниях для считывания и записи значений. Следующая разметка будет обработана:

**HTML**

<!DOCTYPE html>

<html lang="en">

<head>

<title>About Contoso</title>

<meta name="description" content="Let us tell you about Contoso's philosophy and mission.">

...

**Сводка различий между ViewData и ViewBag**

* ViewData
  + Является производным от [ViewDataDictionary](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.viewdatadictionary), поэтому имеет свойства словаря, которые могут быть полезны, например ContainsKey, Add, Remove и Clear.
  + Ключи в словаре представляют собой строки, поэтому пробел допустим. Пример: ViewData["Some Key With Whitespace"]
  + Любой тип, кроме string, должен быть приведен в представлении так, чтобы он мог использоваться в ViewData.
* ViewBag
  + Является производным от [DynamicViewData](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.internal.dynamicviewdata), поэтому позволяет создавать динамические свойства с помощью точечной нотации (@ViewBag.SomeKey = <value or object>); приведение не требуется. Синтаксис свойства ViewBag позволяет быстрее добавлять его в контроллеры и представления.
  + Проще проверять значение NULL. Пример: @ViewBag.Person?.Name

**Выбор между ViewData и ViewBag**

Свойства ViewData и ViewBag в равной степени подходят для передачи небольших объемов данных между контроллерами и представлениями. Выбор одного из них зависит от предпочтений. Вы можете сочетать объекты ViewData и ViewBag, однако код будет проще для восприятия и обслуживания при последовательном применении одного подхода. Оба свойства разрешаются динамически во время выполнения и поэтому могут приводить к ошибкам во время выполнения. Некоторые команды разработчиков избегают их использования.

### Динамические представленияDynamic views

Представления, в которых тип модели не объявляется с помощью @model, но в которые вместо этого передается экземпляр модели (например, return View(Address);), могут ссылаться на свойства экземпляра динамически.

**CSHTML**

<address>

@Model.Street<br>

@Model.City, @Model.State @Model.PostalCode<br>

<abbr title="Phone">P:</abbr> 425.555.0100

</address>

Эта возможность повышает гибкость, но не обеспечивает защиту компиляции или функцию IntelliSense. Если свойство не существует, создание веб-страницы завершается сбоем во время выполнения.