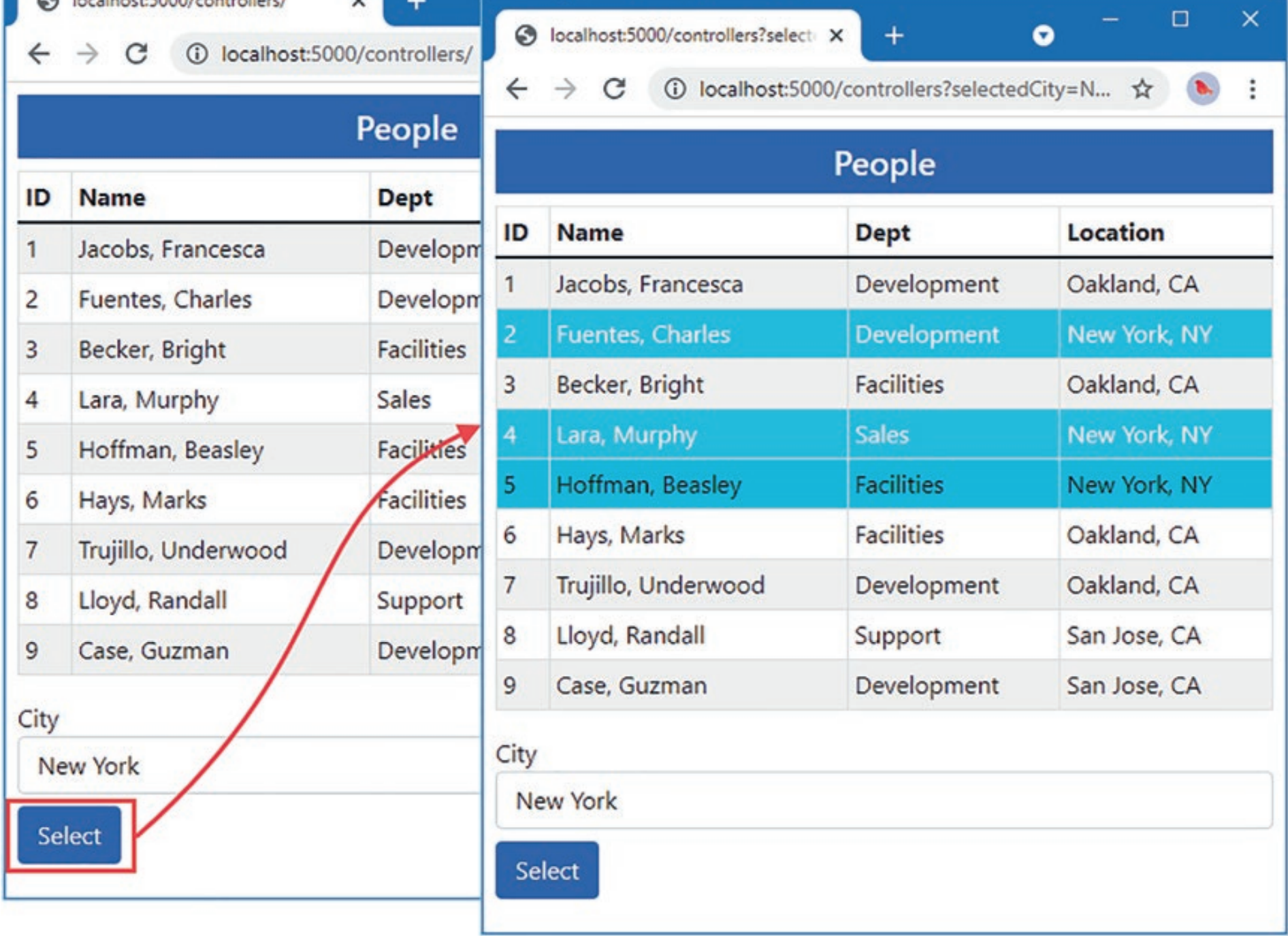
**Глава 33**

**Использование Blazor Server, часть 1.**

Blazor добавляет интерактивности клиентской части веб приложения. Имеется две разновидности Blazor и в этой части сфокусируемся на Blazor Server.

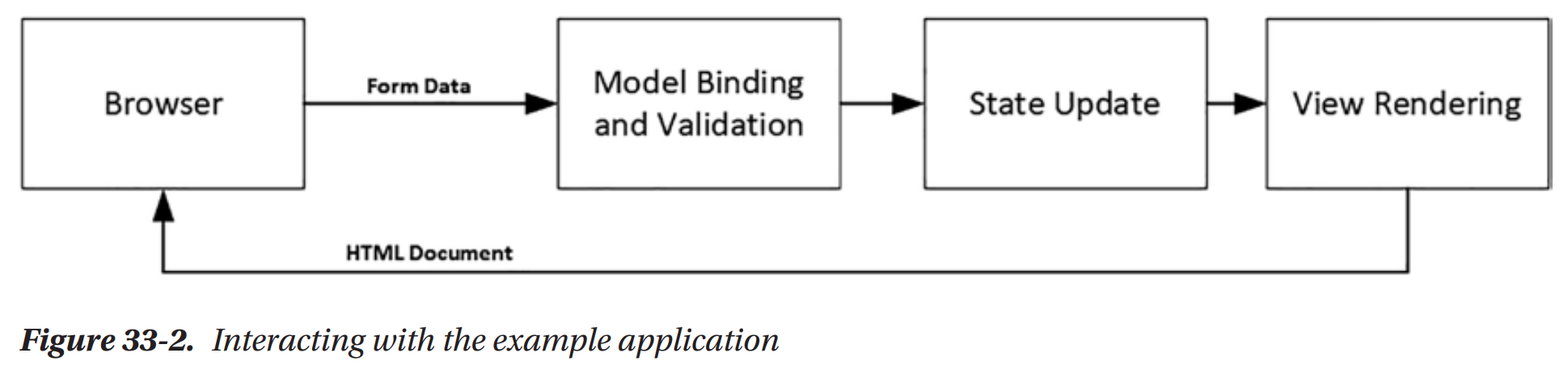
**Подготовка для этой главы**

Подготовка не требуется, все уже настроено. При запуске приложения и переходе по URL <http://localhost:5000/controllers> мы видим:



**Понимание Blazor Server**

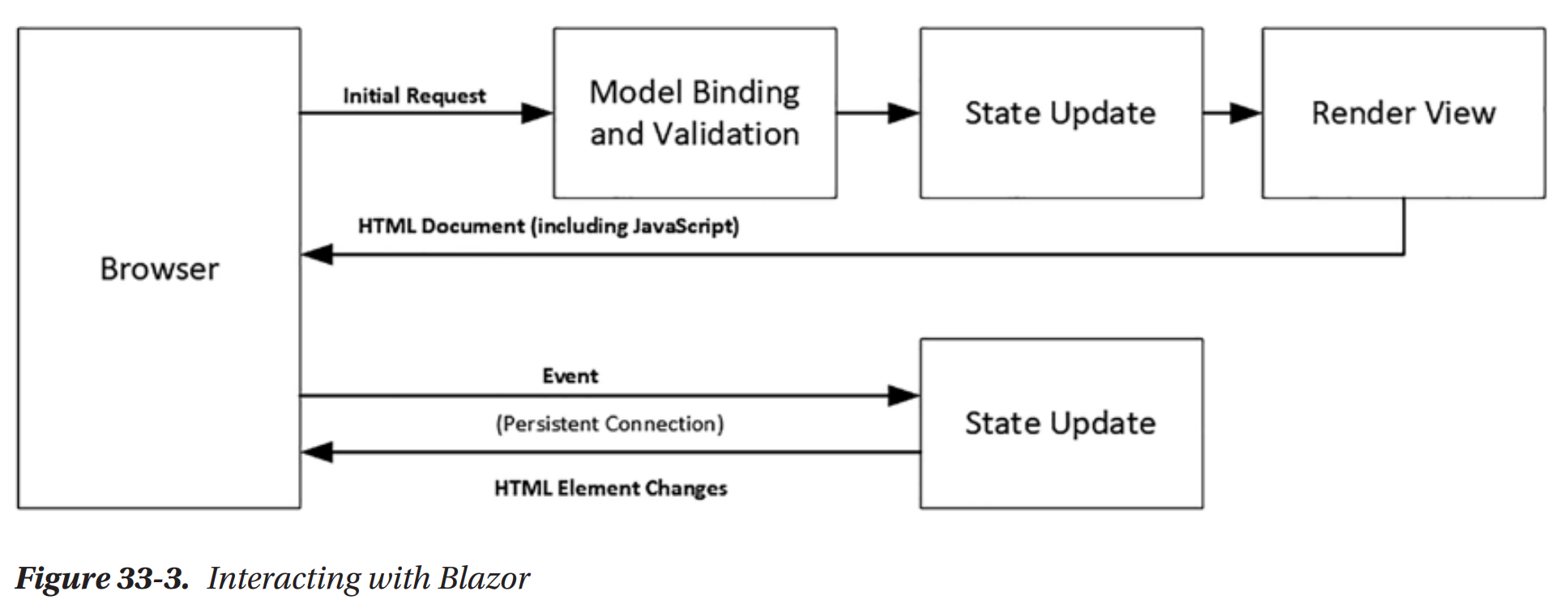
Рассмотрим, что происходит, когда вы выбираете город и кликаете кнопку Select на вышеуказанных изображениях. Браузер отправляет запрос HTTP GET, который выполняет submit, полученный либо методом действия или методом обработчика, в зависимости от того, что использовалось контроллер или Razor Page. Методом действия или метод обработчика рендерит это вью и отправляет новый HTML-документ, отражающий выбор, в браузер.



Этот цикл эффективен, но может быть и не эффективным. Каждый раз, когда нажимается кнопка отправить, браузер отправляет новый запрос HTTP к ASP.NET Core. Каждый запрос содержит полный набор HTTP заголовков, описывающих запрос и тип ответов, которые браузер хотел бы получить. В свой ответ сервер включает HTTP заголовки, описывающие ответ, и полный HTML документ для отображения браузером.

Объем данных, отправляемых примером приложения, составляет около 3 КБ в этом приложении, и почти все данные дублируются между запросами. Браузер только хотел сказать серверу какой город был выбран, а сервер только хотел пометить выбранные строки таблицы, однако, HTTP запрос автономен и браузер должен парсить полный HTML документ каждый раз. Корневая проблема в том, что каждый акт взаимодействия пользователя с приложением повторяет данные действия раз за разом.

Blazor действует по другому. JavaScript библиотека включена в HTML документ, отправляемый браузеру. Когда JavaScript код выполняется, это открывает HTTP соединение обратно к серверу и оставляет его открытым для взаимодействия с пользователем. Когда пользователь выбирает значение, используя элемент select, например, детали выбора отправляются серверу, который отвечает только изменениями для применения в существующем на клиенте HTML.



Постоянное HTTP соединение минимизирует задержку и ответ только с различиями уменьшает объем данных, передаваемых между браузером и сервером.

**Понимание преимуществ Blazor Server**

Наибольшая привлекательность Blazor Server в том, что он основывается на Razor Pages, написанных на C#. Это означает, что вы можете повышать эффективность и отзывчивость без необходимости изучать какой либо другой фреймворк, такой как Angular или React, и другой язык, такой как JS и TS. Blazor мило интегрирован в ASP.NET Core.

**Понимание недостатков Blazor Server**

Blazor требует наличия современных браузеров для установки и поддержания постоянного HTTP соединения. И из за необходимости в таком соединении приложения использующие Blazor перестают работать, когда соединение пропадает, это делает их неподходящими для использования offline, где нельзя полагаться на подключение или где соединение медленное. Эти проблемы решаются посредством Blazor WebAssembly, который будет описан далее, но, как я объясню, он имеет собственный набор ограничений.

**Выбор между Blazor Server и Angular/React/Vue.js**

Если есть возможность изучить один из данных фреймворков, то лучше сделать выбор в их пользу.

**Начинаем Blazor**

В последующих секциях я сконфигурирую приложение для поддержки Blazor и пересоздам функциональность, предоставленную контроллерами и Razor Page.

**Конфигурация ASP.NET Core для Blazor Server**

Первым шагом добавим сервисы и middleware в Program.cs.





Hub в методе MapBlazorHub полагается на SignalR, являющейся частью ASP.NET Core, поддерживающей постоянное HTTP соединение.

**Добавление Blazor JavaScript File в Layout**

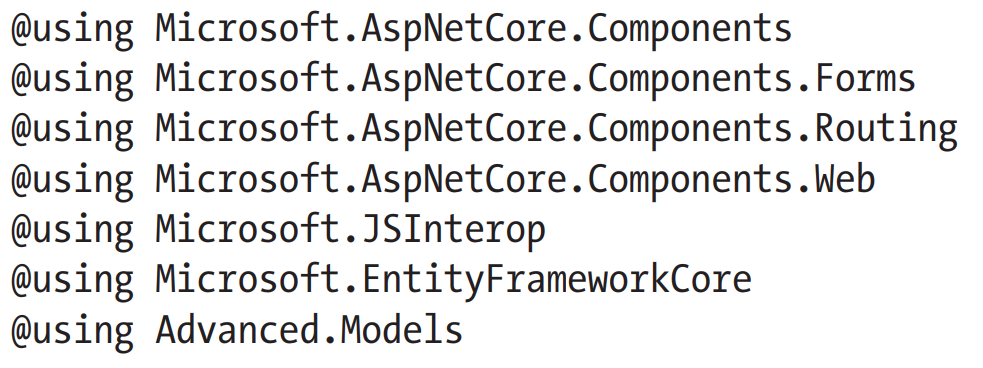
Blazor полагается на JavaScript код для коммуникации с сервером. Добавим элементы в файл лейаута, используемого контроллерами (\_Layout.cshtml в папке Views/Shared).



Элемент script устанавливает имя JavaScript файла и запросы на него перехватываются middleware, добавленным в пайплайн (app.MapBlazorHub();), поэтому не требуется дополнительный пакет для добавления JS кода в проект. Элемент base также должен быть добавлен для указания корневого URL приложения. Такие же элементы необходимо добавить в лейаут, используемый Razor Pages.

**Создание файла импорта Blazor**

Blazor требует свой собственный файл с импортами для указания пространств имен им используемых. Легко забыть добавить его в проект, но без него Blazor молча упадет. Добавим файл \_Imports.razor в корень проекта.

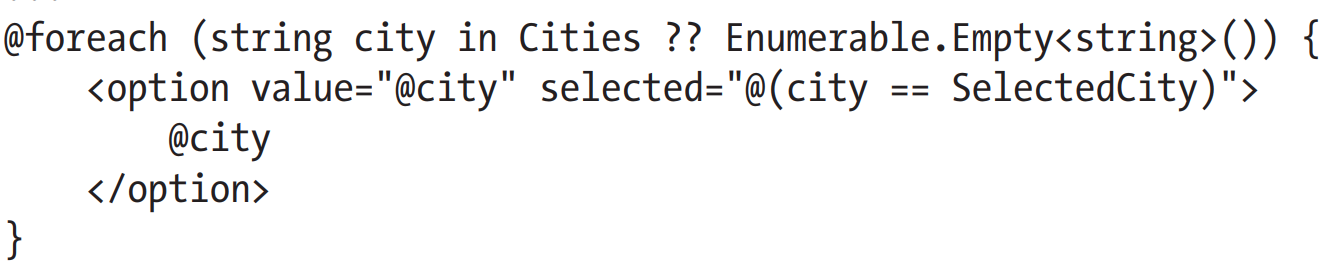


Первое выражение @using для пространств имен, требуемых Blazor, последние два выражения вставлены для удобства, потому что они позволяют использовать EF Core и классы в пространстве имен Models.

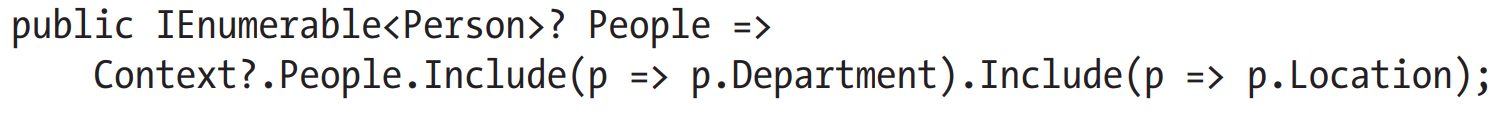
**Создание компонента Razor**

Имеется конфликт в терминологии: технология называется Blazor, но ключевой строительный блок называется Razor Component. Razor Components определяются в файлах с расширением .razor, названия которых должны начинаться с заглавной буквы. Компонент может быть определен где угодно в проекте, но они обычно группируются вместе для органичности проекта. Создадим папку Blazor в корне проекта и добавим в нее файл PeopleList.razor.

Razor Components похожи на Razor Pages. Вью секция полагается на Razor функциональность, с выражениями помеченными @ для вставки значений данных в компонент HTML или для генерации элементов для объектов в последовательности:



Хотя Razor Components выглядят знакомыми, имеется несколько важных различий. Первое то, что нет класса page model и нет выражения @model. Свойства и методы, которые поддерживают компонент HTML, определяются прямо в выражении @code, являющимся аналогом выражения @functions. Для определения свойства, которое, например, будет предоставлять вью секции объект Person, я просто определю свойство People в секции @code:



Из за того, что здесь нет page model класса, здесь нет конструктора, для объявления зависимостей. Вместо этого DI устанавливает значения свойств, которые были декорированы атрибутом [Inject].

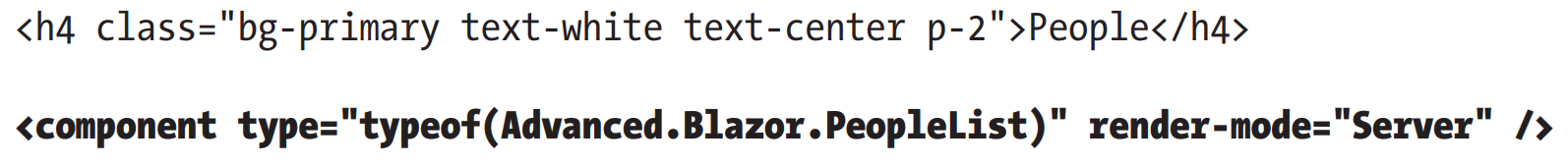
Наиболее значительное различие заключается в использовании специального атрибута в элементе select:



Этот Blazor атрибут создает привязку данных между значением элемента select и свойством SelectedCity, определенным в секции @code. Значение свойства SelectedCity будет изменено когда пользователь изменит значение элемента select.

**Использование Razor Component**

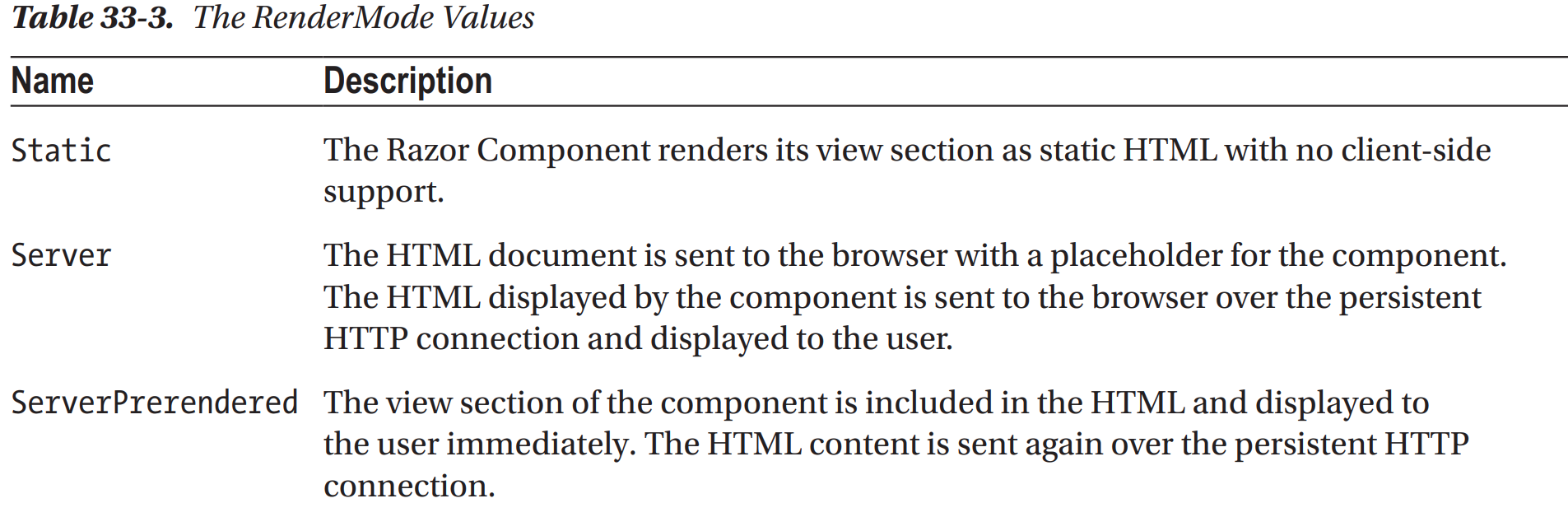
Razor Components доставляются в браузер как часть Razor Page или controller view. Изменим файл Index.cshtml в папке Views/Home:



Razor Components применяются используя элемент component, являющийся тег хелпером. Элемент component конфигурируется используя атрибуты type и render-mode. Атрибут type используется для установки Razor Component. Razor Components компилируются в классы такие же как контроллер вью или Razor Pages. Компонент PeopleList, определен в папке Blazor проекта Advanced, поэтому type будет Advanced.Blazor.PeopleList:



Атрибут render-mode используется для выбора того, какой контент производится компонентом, используя значение перечисления RenderMode, указанным в таблице:



Для большинства приложений опция Server является хорошим выбором. ServerPrerendered включает статическую отрисовку Razor Component’s вью секцию в HTML документе, отправленном браузеру. Это действует как содержимое-заполнитель, чтобы пользователю не показывалось пустое окно браузера, пока выполняется JavaScript код загружается и выполняется. Как только постоянное HTTP соединение установлено, содержимое-заполнитель удаляется и заменяется динамической версией, отправленной Blazorом. Идея показа статического контента пользователю хорошая, но она может сбивать с толку потому что HTML элементы не подключены к серверной части приложения и некоторые взаимодействия от пользователя либо не работают, либо будут отменены, когда пребудет контент от сервера.

Чтобы посмотреть Blazer в действии запустим приложение и перейдем по URL http://localhost:5000/ controllers. При использовании Blazor не требуется отправка формы, потому что привязка данных ответит сразу же как только элемент select будет изменен.

Когда вы используете элемент select, значение, которое вы выбрали, отправляется посредством постоянного HTTP соединения на сервер, где обновляется свойство SelectedCity Razor Component’s и перерисовывается HTML контент. В код JS отправляется набор обновлений, который затем обновляет таблицу. Razor Components могут также использоваться в Razor Pages. Добавим Razor Page с наименованием Blazor.cshtml в папку Pages.

Дополнительный JS код, содержащийся в Blazor.cshtml, помогает продемонстрировать, что только изменения отправляются, а не весь HTML документ. Перезапустим приложение, перейдем по URL http:// localhost:5000/pages/blazor. Кликнем на кнопку Mark Elements и ячейки в ID колонке будут изменены, показывая различающийся контент и бордер. Затем используем элемент select для выбора другого города и вы увидите, что элементы в таблице были модифицированы без удаления.

**Понимание сообщения Blazor connection**

Когда вы остановите приложение ASP.NET Core, вы увидите сообщение об ошибке в окне браузера, которое сообщает о потере соединения с сервером и предупреждает пользователя от взаимодействия с демонстрируемым компонентом. Blazor будет пытаться переподключиться и продолжить с того места, где он остановился, когда произошло отключение из-за временных проблем с сетью, но это может быть не возможно сделать, когда сервер остановлен или перегружается, потому что данные контекста соединения потеряны. Вам придется явно запросить новый URL.

В сообщении о подключении есть ссылка для перезагрузки по умолчанию, но она ведет на дефолтный URL сайта, что бесполезно для этой книги, где я направляю вас на определенные URL-адреса, чтобы увидеть эффект примеров.

**Понимание базового функционала Razor Component**

Я продемонстрировал как Blazor может быть использован и как он работает, пришло время вернуться к базе и представить функциональность, предлагаемую компонентами Blazor. Хотя примеры из предыдущей секции демонстрируют как стандартный функционал ASP.NET Core может быть переделан с использованием Blazor, имеется более обширный спектр доступной функциональности.

**Понимание событий Blazor и привязки данных**

События позволяют компоненту Razor отвечать на пользовательские действия, и Blazor использует постоянное HTTP содединение для отправки деталей события серверу, где они обрабатываются. Для того, чтобы увидеть события Blazor в действии добавим компонент Razor с именем Events.razor в папку Blazor.

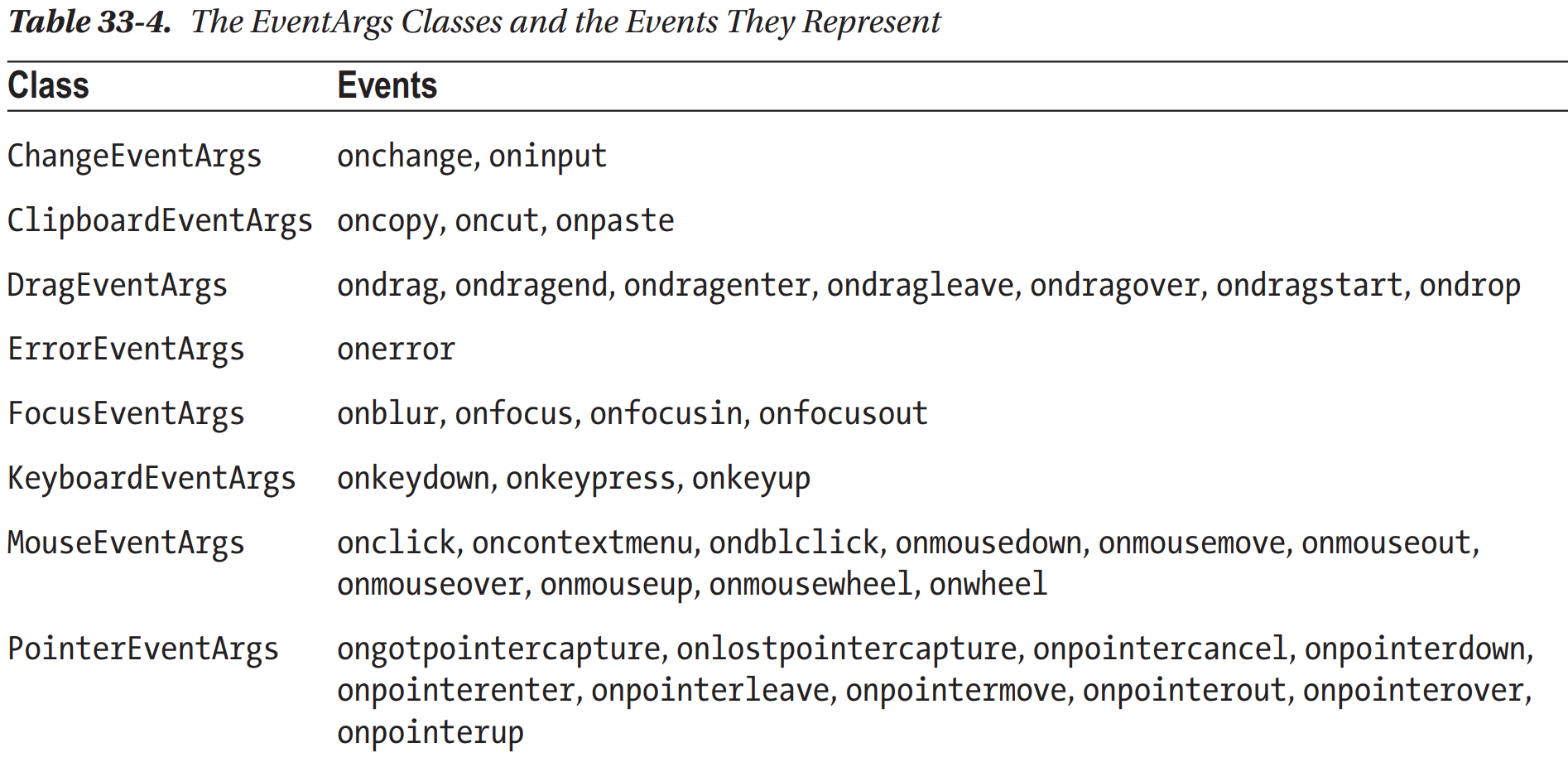
Мы регистрируем обработчик события путем добавления атрибута с наименованием @on с последующим именем события (click) к HTML элементу. В примере я установил обработчик для события клик, генерируемого элементом кнопки:

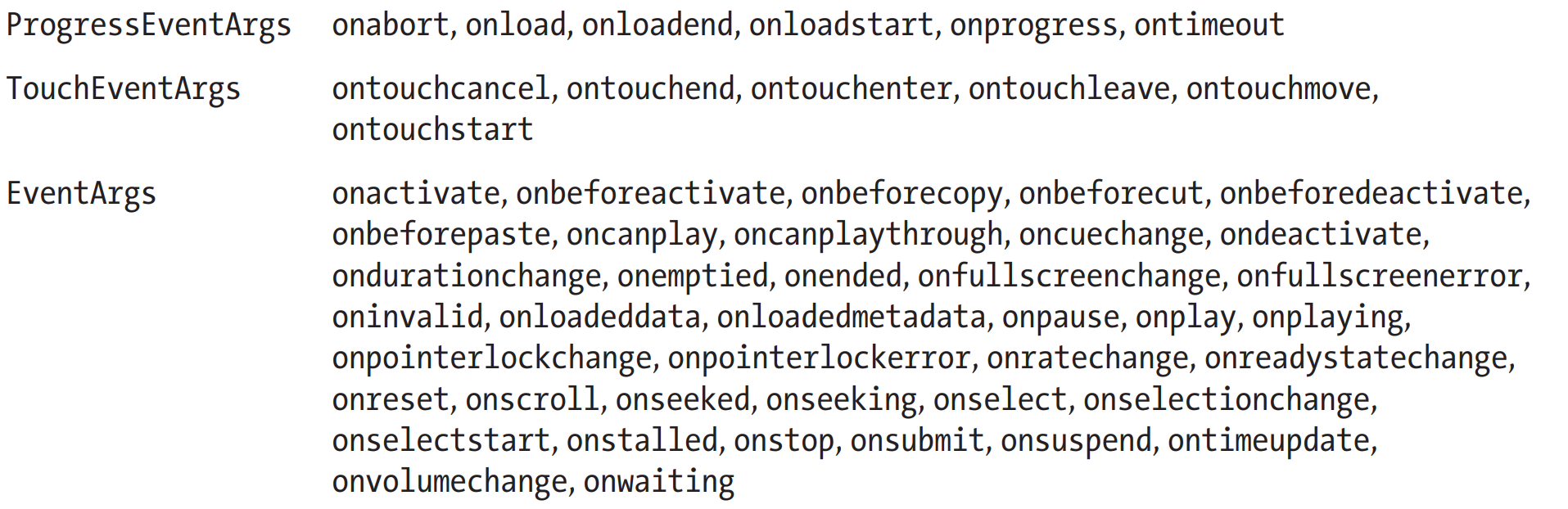


Значение, присваиваемое данному атрибуту это имя метода, который выполняется когда событие запущено. Данный метод может быть определен с опциональным параметром, который либо является экземпляром класса EventArgs, либо экземпляром класса, наследуемого от EventArgs, который предоставляет доп. информацию о событии.

Для события клик метод обработчика принимает объект класса MouseEventArgs, который предоставляет дополнительные детали, такие как экранные координаты клика.

В нижеприведенной таблице описаны классы, наследуемые от класса EventArgs и события, для которых они используются:





Blazor JavaScript код получает запущенное событие и отправляет на сервер посредством постоянного HTTP соединения. Метод обработчика события выполняется и состояние компонента обновляется. Любые изменения в содержимом, производимом разделом представления компонента будут отправлены обратно JavaScript коду, который обновляет контент, отправляемый браузеру.

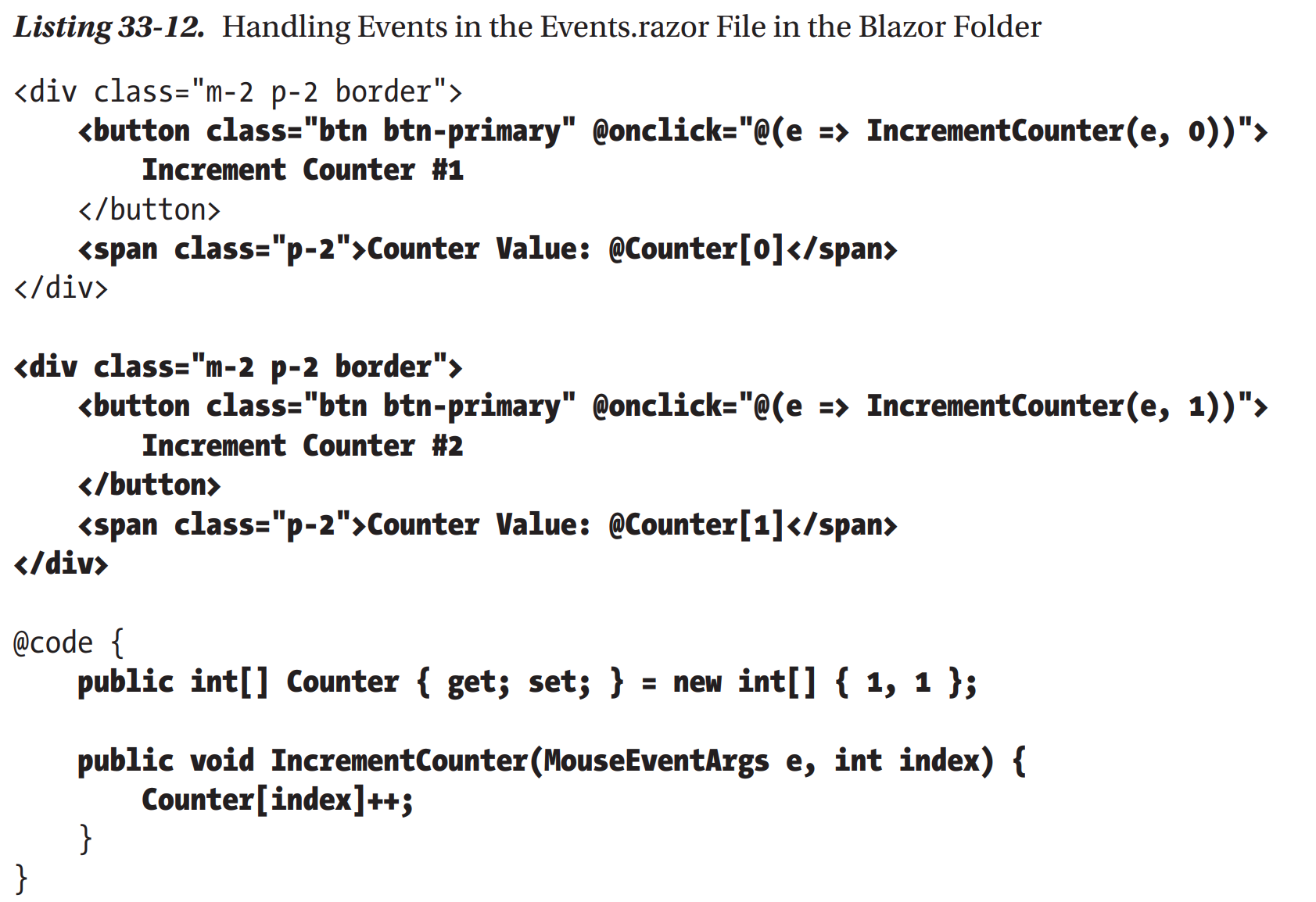
В примере событие клика будет обработано обработчиком события IncrementCounter, который изменяет значение свойства Counter. Значение свойства Counter включается в HTML-код, отображаемый компонентом, поэтому Blazor отправляет изменения в браузер, чтобы код JavaScript мог обновить HTML элементы, отображаемые пользователю. Для демонстрации компонента Events заменим контент в файле Blazor.cshtml в папке Pages.



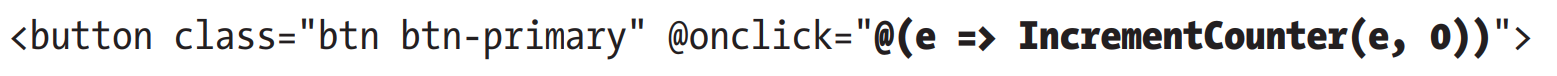
Данный листинг изменяет атрибут type, удаляет кастомный JavaScript и кнопку, которые были использованы в прошлом примере. Перезапустим приложение и перейдем по URL <http://localhost:5000/pages/blazor>, чтобы увидеть новый компонент.

**Обработка событий множества элементов**

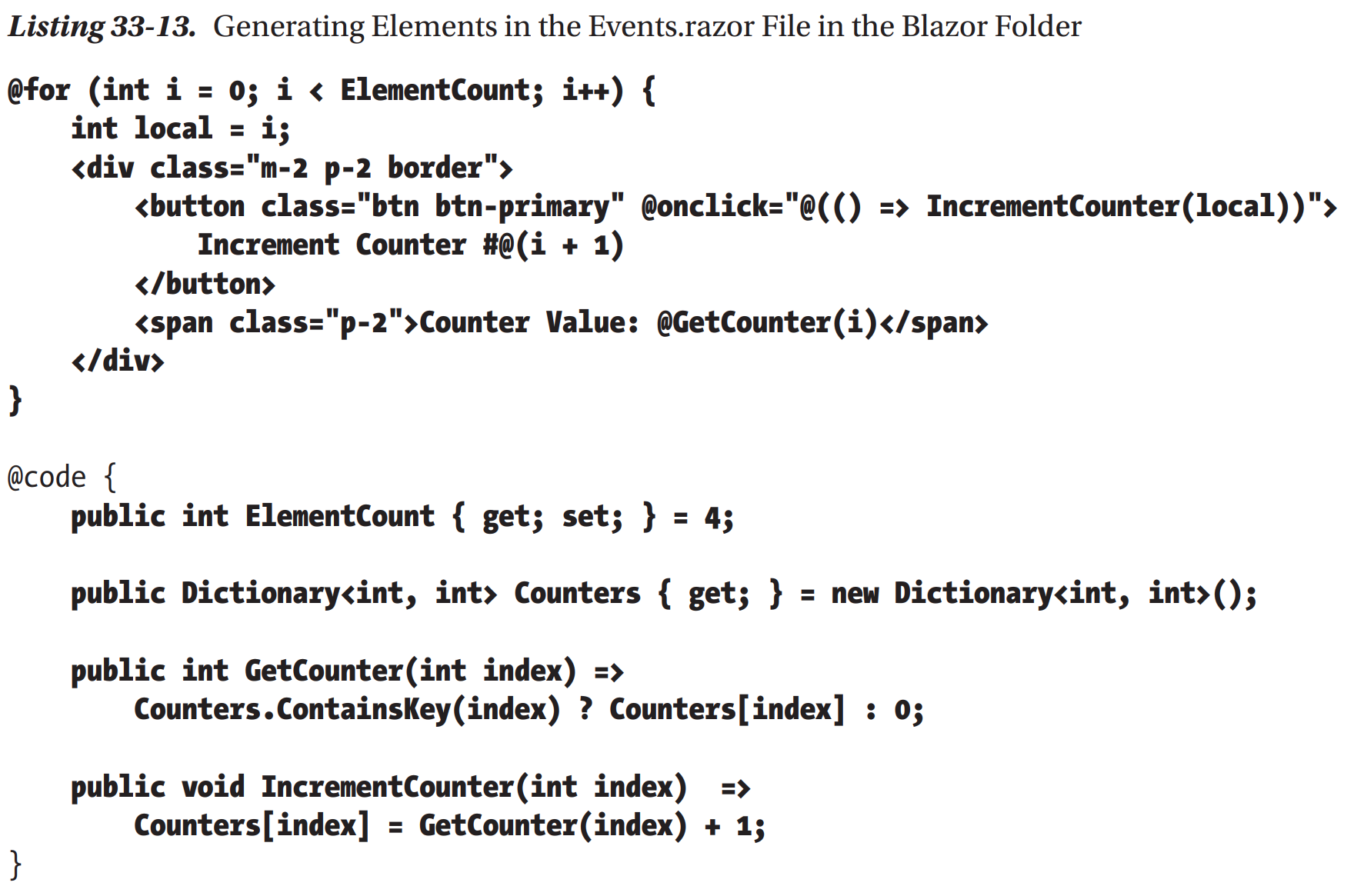
Для того, чтобы избежать дублирования кода, события от множества элементов могут быть получены с помощью одного обработчика:



Атрибуты события Blazor могут использоваться с лямбда выражениями, получающими экземпляр класса EventArgs и выполняющими метод обработчика с дополнительными аргументами. В данном примере я добавил параметр index в метод IncrementCounter, который используется для определения какой из счетчиков должен быть обновлен. Значение для аргумента определено в атрибуте @onclick:



Данная техника может быть использована когда элементы генерируются программно, как показано в листинге:

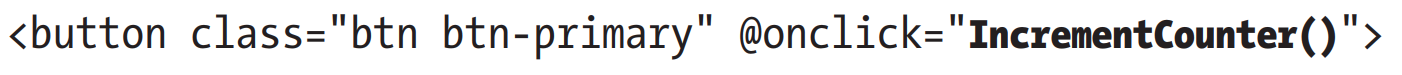


В этом примере я использовал выражение @for для генерации элементов и использовал переменную цикла как аргумент метода обработчика, а также удалил параметр EventArgs, так как он в данном случае не используется.

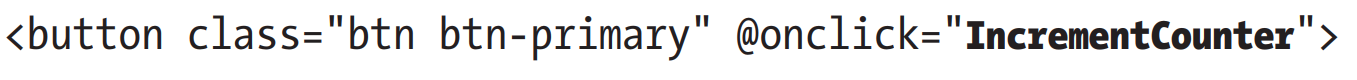
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Как избежать ловушки имени метода обработчика события**

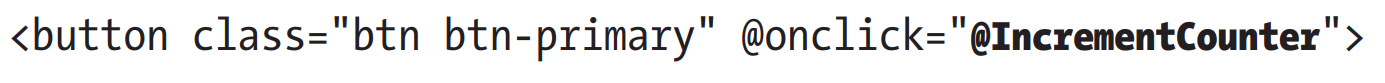
Наиболее распространенной ошибкой при указании метода обработчика событий является включение круглых скобок:



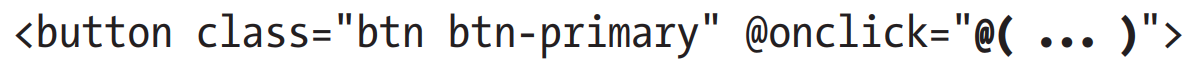
Когда вы указываете метода обработчика событий, вы должны просто указать имя метода:



Вы также можете указать данный метод как выражение Razor:



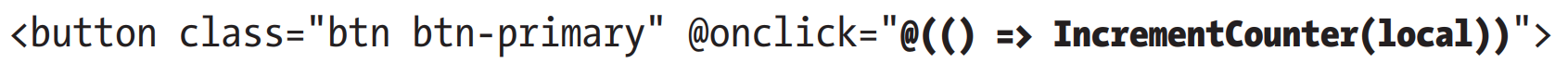
Некоторые разработчики находят вышеуказанный подход более читаемым, но результат будет одним и тем же. Для использования лямбда выражений применяется другой набор правил, который должен быть определен как выражение Razor:



Внутри выражения Razor лямбда выражение определяется также как и в C# классе, что означает возможность использования параметров метода, за которыми следует стрелка и тело функции:



Если вам не нужно использовать объект класса EventArgs, вы можете упустить параметр е:



Важный момент для понимания обработчиков событий это то, что лямбда выражение аргумента @onclick не выполняется пока не получит событие клика из браузера. Это означает, что нужно быть осторожным и не использовать переменную цикла I в качестве аргумента метода IncrementCounter, потому что она будет всегда иметь конечное значение цикла (4 в данном примере). Вместо этого вы должны присвоить переменную цикла в локальную переменную:



**Обработка событий без метода обработчика**

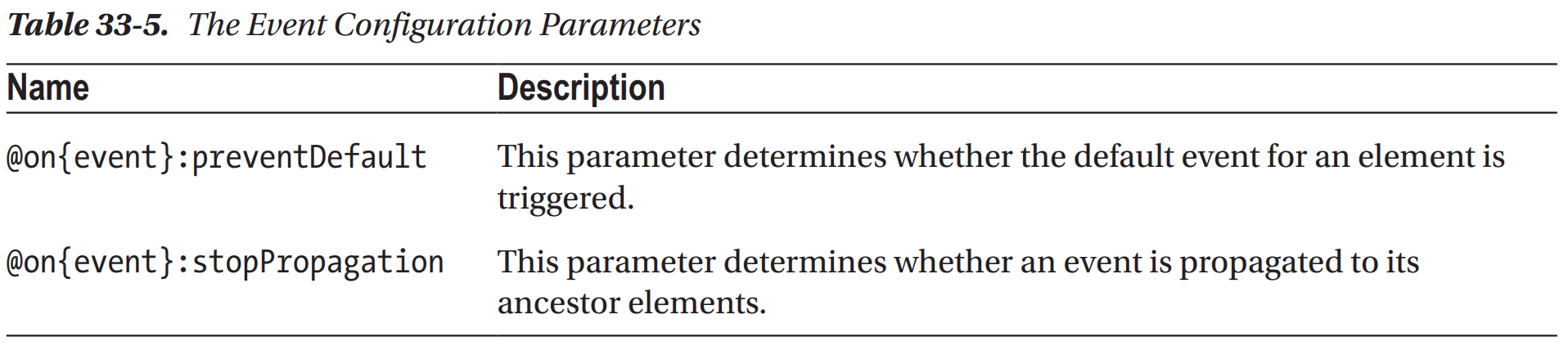
Простые события могут быть обработаны прямо в лямбда выражении, без использования метода обработчика:



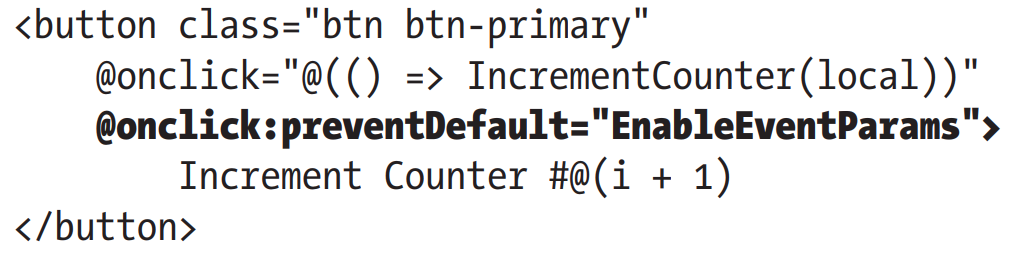
Комплексные обработчики следует определять как методы, но данный подход более подходит для простых обработчиков.

**Предотвращение дефолтных событий и распространение событий.**

Blazor предоставляет два атрибута, которые изменяют поведение событий браузера по умолчанию, как описано в таблице:



Это атрибуты, в которых за именем события следует двоеточие, а затем ключевое слово, известное как параметры.





Вышеуказанный пример создает две ситуации, в которых дефолтное поведение событий в браузере создает проблемы. Первая из-за добавления элемента form. По умолчанию кнопка в элементе form отправляет данные формы при клике, даже если представлен атрибут @onclick. Это означает, что когда бы ни была нажата кнопка Increment Counter, браузер отправит форму на сервер, который отправит ответ с контентом, содержащимся в файле Blazor.cshtml, то есть увеличения значения свойства Counter не произойдет.

Вторая проблема продемонстрирована посредством элемента, родитель которого также определяет обработчик события:



**Работа с привязкой данных**

Обработчики событий и выражения Razor могут быть использованы для создания двунаправленных отношений между элементами HTML и значениями C#, что полезно для элементов, позволяющих пользователю вносить изменения, такие как input и select. Для примера добавим компонент Razor с наименованием Bindings.razor в папку Blazor.

Атрибут @onchange в данном примере регистрирует метод UpdateCity как обработчик события change на элементе input. События change описываются с помощью класса ChangeEventArgs, который предоставляет свойство Value. Каждый раз, когда получено событие change, свойство City обновляется значением из элемента input.

Атрибут value элемента input создает отношение в обратном направлении, так что когда изменяется значение свойства City, также изменяется значение атрибута value.

Для применения нового Razor компонента, изменим атрибут component в файле Blazor.cshtml:



Двунаправленное отношение подразумевает, что событие change может быть выражено как привязка данных, что позволяет атрибуту value и событию change быть заданными единственным атрибутом @bind="City":

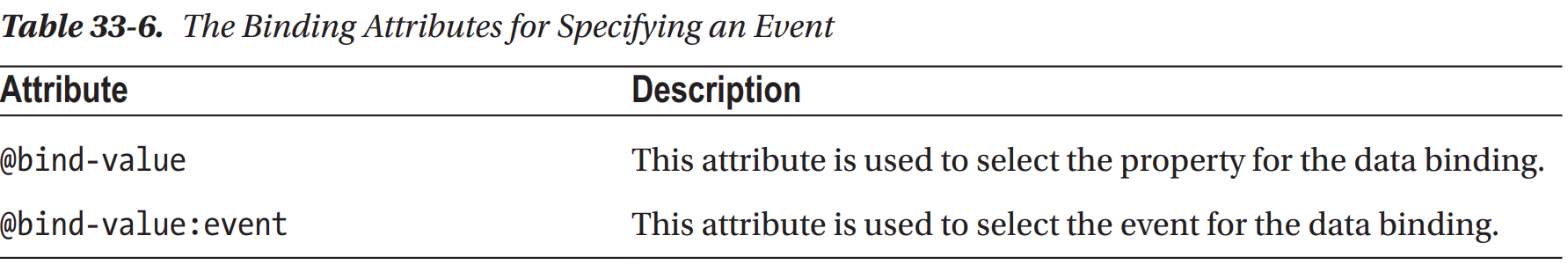


При этом избавившись от метода обработчика события change.

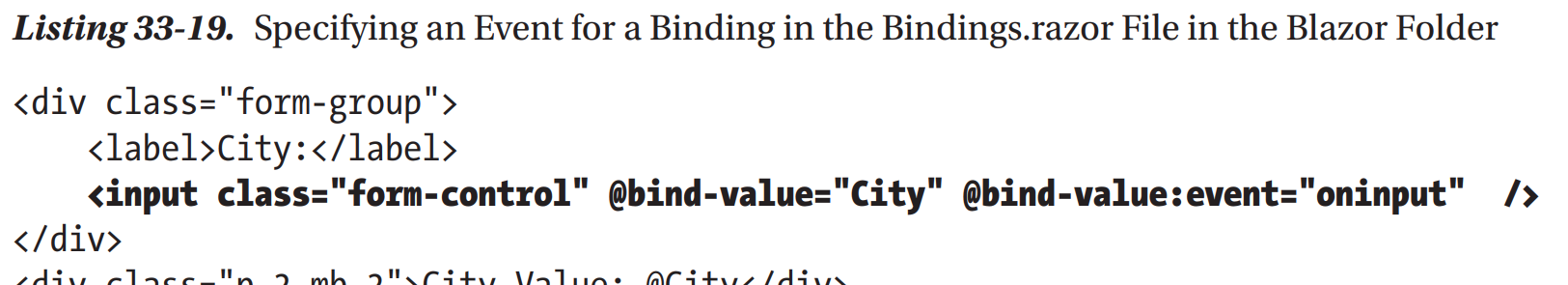
Атрибут @bind используется для задания свойства, которое будет обновляться когда фиксируется событие change и что атрибут value будет изменен.

**Изменение события привязки**

По умолчанию в привязках используется событие change, что обеспечивает разумную скорость отклика для пользователя, не требуя слишком много обновлений от сервера. Событие, используемое для привязки может быть изменено путем использования атрибутов, указанных в таблице:



Эти атрибуты используются вместо атрибута @bind, но могут быть использованы только с событиями, представленными классом ChangeEventArgs. Это означает, что только события onchange и oninput могут быть использованы, по крайней мере в текущем релизе.



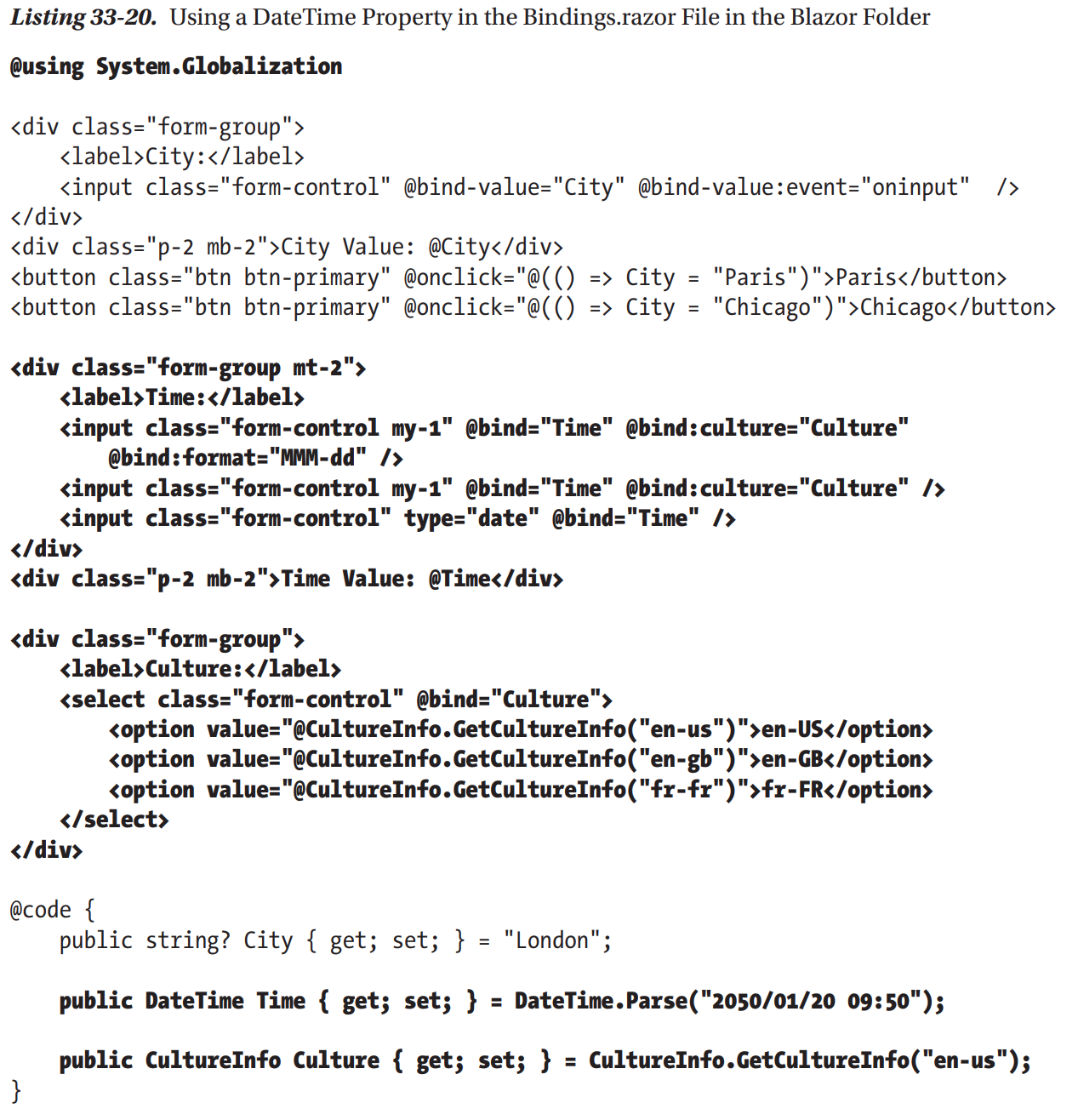
Комбинация атрибутов, определенная в вышеуказанном листинге, создает привязку для свойства City, когда фиксируется событие oninput, которое происходит каждый раз, когда вносятся изменение в поле input, а не тогда, когда поле input теряет фокус.

**Создание привязки DateTime**

Blazor имеет специальную поддержку для создания привязки для свойства DateTime, позволяя им быть выраженным используя специальную культуру или формат строки. Это функциональность может быть применена используя параметры, указанные в таблице:



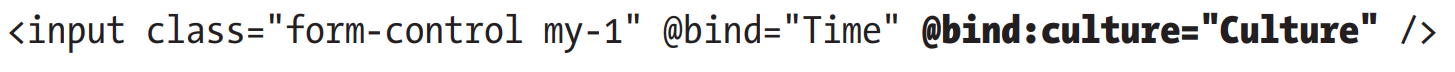
**Примечание**: Если вы использовали атрибуты @bind-value и @bind-value:event для выбора события, вместо этого вы должны использовать параметры @bind-value:culture и @bind-value:format. Изменим файл Bindings.razor:



Здесь описаны три элемента input, используемые для показа одного и того же значения свойства DateTime, два из которых были сконфигурированы с использованием атрибутов из вышеуказанной таблицы. Первый элемент был сконфигурирован с culture и format string:



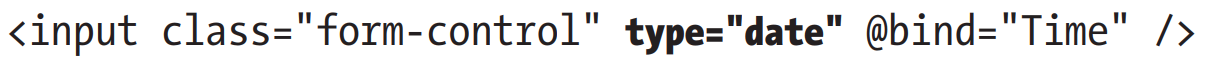
Свойство DateTime показано с использованием культуры, выбранной из элемента select и с format string, дающим сокращенное название месяца и числовую дату. Второй элемент input задается только с культурой, означающей, что будет использоваться строка форматирования по умолчанию.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАЗРЕШЕНИЕ ФОРМАТИРОВАНИЯ ДАТЫ В БРАУЗЕРЕ**

Обратите внимание, что значение, отображаемое третьим элементом input в вышеуказанном листинге не изменяется не смотря на выбранную локализацию. Этот элемент input не имеет атрибутов, указанных в таблице 33-7, но имеет собственный атрибут type, установленный в date:



Вы не должны задавать культуру или формат строки (culture, format string), когда устанавливаете атрибут type в date, datetime-local, month, или time, потому что Blazor будет автоматически значение даты в culture-neutral формат, который браузер транслирует на язык пользователя.

**Использование файла класса для определения Razor компонента**

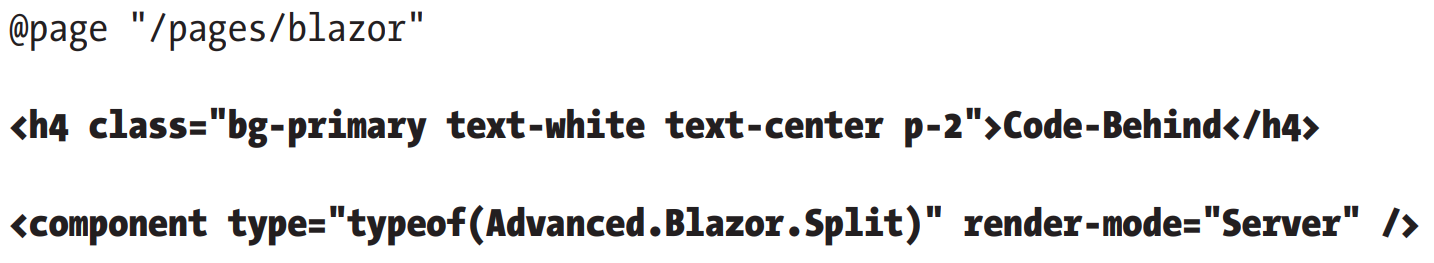
Если вы не любите смесь кода C# и разметки, поддерживаемую компонентом Razor, вы можете использовать класс C#, для определения части или всего компонента.

**Использование класса Code-Behind**

Секция @code компонента Razor может быть определена в отдельном классе, который принято называть Code-Behind. Класс Code-Behind для компонента Razor определяется как partial класс с таким же именем как и у компонента, которому они предоставляют код.

Добавим Razor компонент с именем Split.razor в папку Blazor. Он будет содержать только HTML контент и выражения Razor и отображает список имен, которые ожидается получить из свойства Names. Для предоставления компоненту его кода добавим класс Split.razor.cs в папку Blazor и используем его, чтобы определить partial класс. Partial класс должен быть определен в том же пространстве имен, что и его Razor компонент и иметь такое же имя. Классы Code-behind не имеют конструктора и получают зависимости используя атрибут Inject.

Для применения нового компонента изменим файл Blazor.cshtml:

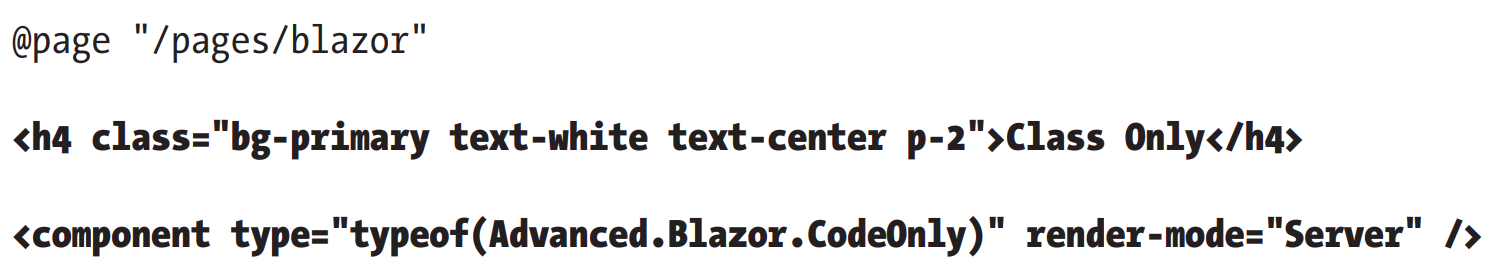


**Определение класса Razor компонента**

Компонент Razor может быть определен полностью в классе, хотя это может быть менее выразительно, чем использование выражений Razor. Для наглядности добавим класс CodeOnly.cs в папку Blazor.

Базовым классом для компонента является ComponentBase. Контент, который обычно выражается путем HTML элементов создается путем переопределения метода BuildRenderTree и использования параметра RenderTreeBuilder. Создание контента может быть неуклюжим, потому что каждый элемент создается и конфигурируется с использованием множества строк кода, каждая из которых должна иметь последовательный номер, который компилятор использует для сопоставления кода и содержимого. Метод OpenElement начинает новый элемент, который затем конфигурируется с помощью методов AddContent и AddAttribute и затем завершается методом CloseElement. Доступны также вся функциональность, доступная в обычных компонентах Razor, включая события и привязки данных, которые устанавливаются путем добавления атрибутов к элементам, так же, как если бы они были определены буквально в файле .razor. Компонент, определенный в классе CodeOnly.cs выводит список отсортированных имен, в котором порядок сортировки изменяется путем нажатия кнопки.

Применим данный компонент в файле Blazor.cshtml:



**Итог**

В этой главе я представил Blazor Server, объяснил проблемы, которые он решает, и описал преимущества и недостатки этой технологии. Я показал как конфигурировать приложение ASP.NET Core для возможности использования Blazor Server и показал базовую функциональность, используемую в компонентах Razor, являющимся строительными блоками Blazor. В следующей части я продолжу описывать функциональность предоставляемую Blazor(ом).