

ためしてみよう ASP. NET Core Blazor

T. Matsumori



Blazor とは



- Microsoft の ASP.NET の一番新しいやつ。
 - ASP.NET については、複雑なのでまた別機会。
 - Blazor には3種類ある。
MSのPHPみたいなやつ
- ① Blazor WebAssembly (SPA: シングルページアプリ)
 - ② Blazor Server (SignalR。シンククライアント)
 - ③ Blazor United (①②の統合。未リリース)
- 今回は①のBlazor WebAssemblyを取り上げる。

Blazor WebAssembly

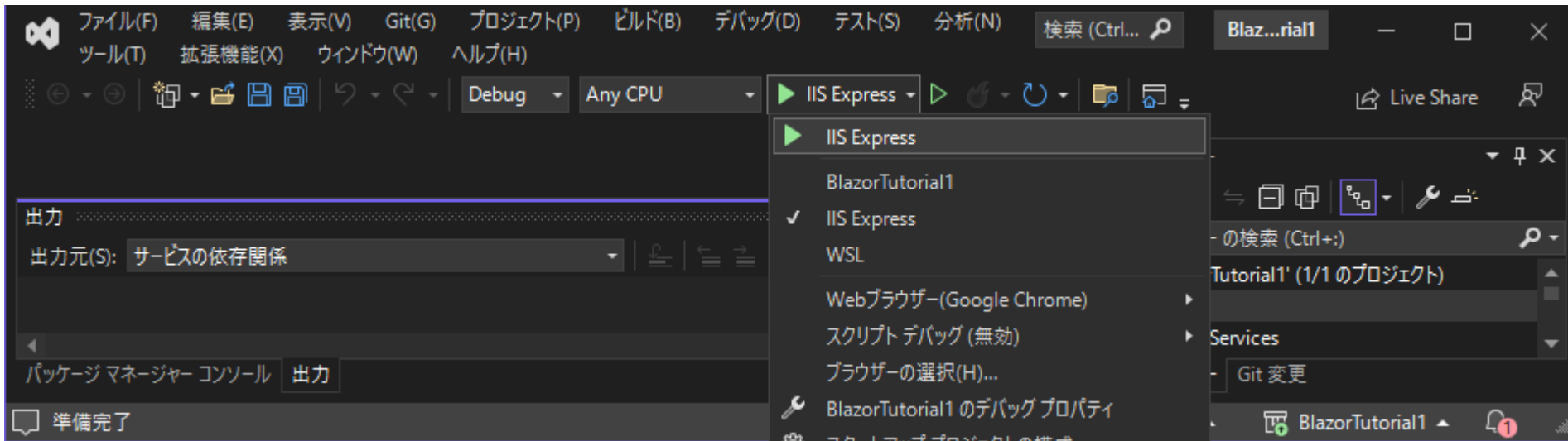
- WebAssembly とはブラウザがバイナリを受け取って実行できるようになったもの。JS代替というより別の選択肢
- 2019年にはW3C勧告「WebAssembly Core Specification」が策定。WebAssemblyは正式なウェブ標準に認定。
- 主要ブラウザでは既に対応済み。サーバーからバイナリファイルを受け取り、ブラウザでクライアント側実行。ブラウザでC#が動く！
- 言語に依存しない。いろいろな言語からWebAssemblyのバイナリファイルが作れる。

教材

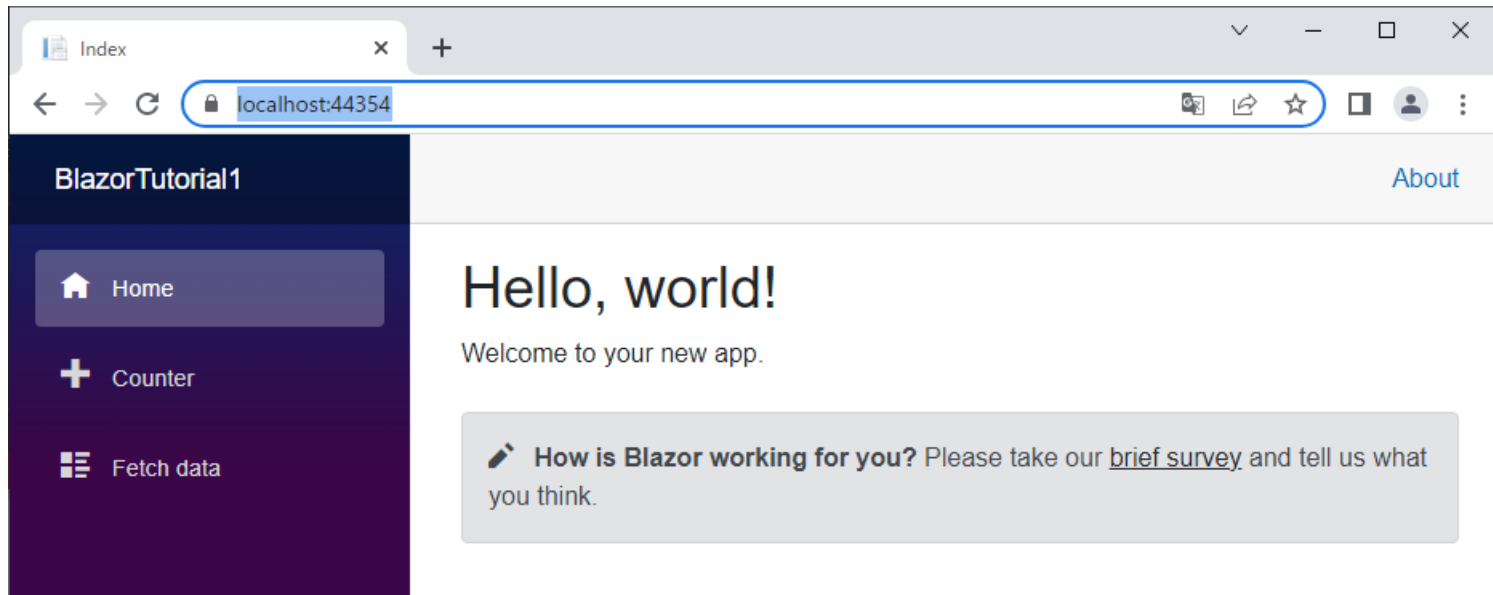
- CodeZine に良い連載があったため教材に。
 - ASP.NET Core Blazorチュートリアル
 - <https://codezine.jp/article/corner/840>
-
- 連載は5回(2020～21年)。
今回はそのうち、第一回を進める。
-
- ① C#でSPAが実現できる、Blazor WebAssembly
のはじめかた

ではさっそく。。。

- Page 2 の[プロジェクトの作成]。
- Visual Studio Community 2022 にて Blazor WebAssembly アプリ をクリック。
- ソリューション名は「BlazorTutorial1」。
- コンボで「IIS Express」を指定しビルド。



サンプルが立ち上がります



- ページ遷移の無いSPA:シングルページアプリケーション
- Home, Counter, Fetch data それぞれ機能する。
- 例: AdminLTE (Bootstrapベース)とかでよく見るやつだが。。。

確かにバイナリを読んでいる




Debugging hotkey: Shift+Alt+D (when application has focus)

▼ **blazor** Loaded 9.19 MB resources
This application was built with linking (tree shaking) disabled. Published applications will be significantly smaller.

▼ Loaded 9.19 MB resources from cache

(インデックス)	responseBytes
Microsoft.AspNetCore.Authorization.dll	22943
Microsoft.AspNetCore.Components.dll	92330
Microsoft.AspNetCore.Components.Forms.dll	17912
Microsoft.AspNetCore.Components.Web.dll	54995
Microsoft.AspNetCore.Components.WebAssembly.dll	41961
Microsoft.AspNetCore.Metadata.dll	9978
Microsoft.Extensions.Configuration.dll	18348
Microsoft.Extensions.Configuration.Abstractions.dll	13219

- 最初に 9.19MB DLLしてSPA作っている。
- 二回目以降はキャッシュが効く。



Debugging hotkey: Shift+Alt+D (when application has focus)

▼ **blazor** Loaded 9.19 MB resources
This application was built with linking (tree shaking) disabled. Published applications will be significantly smaller.

▶ Loaded 9.17 MB resources from cache

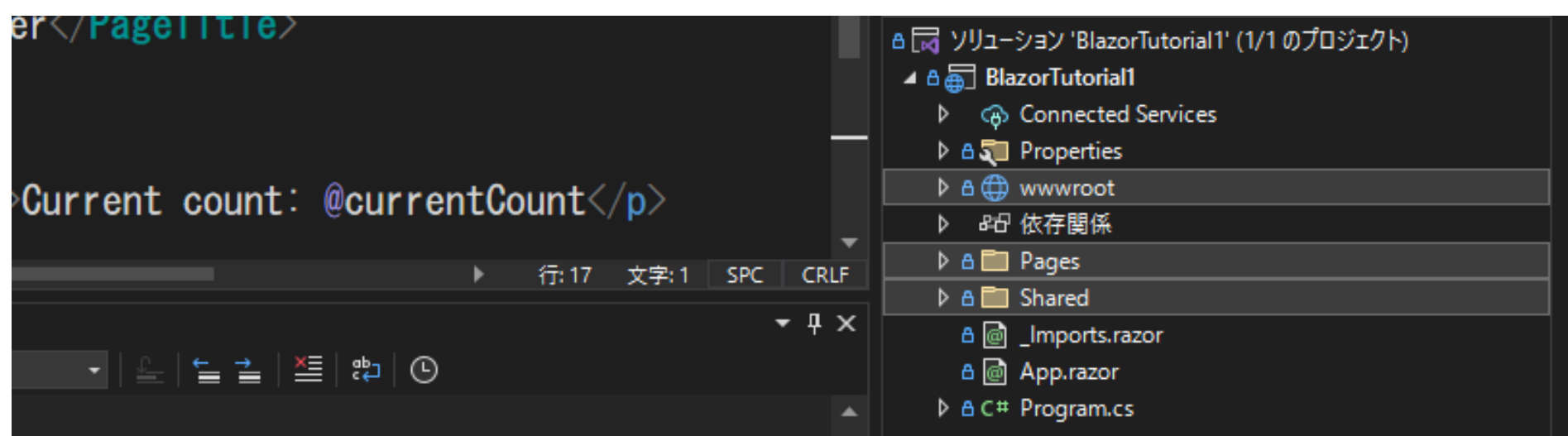
▼ Loaded 0.02 MB resources from network

- ▼ Object 1
 - ▶ BlazorTutorial1.dll: {responseBytes: 8075}
 - ▶ BlazorTutorial1.pdb: {responseBytes: 17177}
 - ▶ [[Prototype]]: Object

変更した差分のDLL

プロジェクトのファイル構成

- 3つのフォルダが重要



- wwwroot: 静的なファイルを配置する
- Pages: 各ページを定義
- Shared: 各ページのメニューを実現する
コンポーネント・子コンテンツのコンポーネント

wwwroot

- wwwroot/index.html が最初に読み込まれる
- index.html では blazor.webassembly.js を呼ぶ。
- これが .NET アセンブリをロードし、ランタイムを初期化して、プロジェクトのコードを実行する。
- index.html の次は App.razor が呼ばれる。
- そこでは MainLayout.razor を呼んでいる。

```
1  <Router AppAssembly="@typeof(App).Assembly">
2  |   <Found Context="routeData">                                // Assembly の型取得
3  |       <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
4  |       <FocusOnNavigate RouteData="@routeData" Selector="h1" />
5  |   </Found>
6  |   <NotFound>                                                  // ↑見つかった場合と
7  |       <PageTitle>Not found</PageTitle>
8  |       <LayoutView Layout="@typeof(MainLayout)">              // ↓見つからなかった場合
9  |           <p role="alert">Sorry, there's nothing at this address.</p>
10 |       </LayoutView>
11 |   </NotFound>
12 </Router>
```

Shared/MainLayout.razor

- メインのレイアウトを提供する HTML の部品が作られる。ただし C# 実行可能。
- NavMenu.razor はサイドメニューを提供する。

The image shows a web browser window displaying a Blazor application. The browser address bar shows `https://localhost:44316`. The application has a dark blue sidebar menu on the left with links: Home, Counter, and Fetch data. The main content area has a white background with the text "Hello, world!" and "Welcome to your new app." Below this is a survey link: "How is Blazor working for you? Please take our [brief survey](#) and tell us what you think."

Overlaid on the browser window is a code editor showing the `MainLayout.razor` file. The code is as follows:

```
@inherits LayoutComponentBase<div class="page"><div class="sidebar"><NavMenu /></div><main><div class="top-row px-4"><a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a></div><article class="content px-4"><@Body></article></main></div>
```

Two yellow arrows point from the code to the browser window. One arrow points from the `<NavMenu />` code to the sidebar menu, which is labeled "NavMenu" in a red box. The other arrow points from the `@Body` code to the main content area, which is labeled "Body プロパティ" in a red box.

MainLayout.razor

<https://codezine.jp/article/detail/13036?p=3>

.razor とは

Blazor = Browser + Razor

- Razor ファイルは C# を直接記述できる HTML。過去の Razor Pages の .cshtml と同一。
- Blazor フレームワークでは拡張子.razorに変更。
- @ を付けると C# を実行できる。



Pages/Counter.razor

```
1  @page "/counter"
2
3  <PageTitle>Counter</PageTitle>
4
5  <h1>Counter</h1>
6
7  <p role="status">Current count: @currentCount</p>
8
9  <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
10
11  @code {
12      private int currentCount = 0;
13
14      private void IncrementCount()
15      {
16          currentCount++;
17      }
18  }
```

Shared ディレクトリ

- Pages で共通するレイアウトを定義するテンプレート。
- MainLayout.razor、NavMenu.razor、また SurveyPrompt.razor が含まれている。

MainLayout.razor

```
@inherits LayoutComponentBase
<div class="page">
  <div class="sidebar">
    <NavMenu />
  </div>
  <main>
    <div class="top-row px-4">
      <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
    </div>
    <article class="content px-4">
      @Body
    </article>
  </main>
</div>
```

← レイアウトとしてテンプレートを定義する
場合は LayoutComponentBase の継承が必要

↑ NavMenu.razor

← Index.razor や Counter.razor、FetchData.razor
サイドバーのクリックでページ(Bodyの中身)変化

Pages ディレクトリ

- Index.razor、Counter.razor、FetchData.razor。
- サイドメニューをクリックして表示される画面。
- 以下は Index.razor。
- SurveyPrompt は Shared にある .razor。
- Title に @DateTime.Now を渡してみた。

Pages/Index.razor

```
1  @page "/"           ← ページのURLを指定。/ は Webサイトのroot
2
3  <PageTitle>Index</PageTitle>
4
5  <h1>Hello, world!</h1>
6
7  Welcome to your new app.
8
9  <!--<SurveyPrompt Title="How is Blazor working for you?" />-->
10 <SurveyPrompt Title="@DateTime.Now.ToString()" />
```

Counter.razor

@currentCount

- HTML側で@を付けて、値を取得している。
- .razor では、ファイル名と同じ名前の C# クラスが作られる。currentCount や IncrementCount はメンバ。

```
1  @page "/counter"
2
3  <PageTitle>Counter</PageTitle>
4
5  <h1>Counter</h1>
6
7  <p role="status">Current count: @currentCount</p>
8
9  <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
10
11  @code {
12      private int currentCount = 0;
13
14      private void IncrementCount()
15      {
16          currentCount++;
17      }
18  }
```

C# のコード
private なメンバ

FetchData.razor

- まず、C#側 (@code{}) にて、WeatherForecast クラスのインスタンスを、forecasts 配列に格納する。
- GetFromJsonAsync にて、json ファイルから自動的にクラスを作っている。

```
39  @code {  
40      private WeatherForecast[]? forecasts;  
41  
42      protected override async Task OnInitializedAsync()  
43      {  
44          forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("sample-data/weather.json");  
45      }  
46  
47      public class WeatherForecast  
48      {  
49          public DateTime Date { get; set; }  
50          public int TemperatureC { get; set; }  
51          public string? Summary { get; set; }  
52          public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);  
53      }  
54  }  
55  
56  
57
```

WeatherForecast
クラス

sample-data/weather.json

The screenshot shows the Visual Studio IDE with the `weather.json` file open in the editor. The file contains a JSON array of five weather forecast objects. A yellow bracket highlights the entire JSON array. To the right of the code, a yellow text box states: **この json より WeatherForecast クラスを自動生成** (Automatically generate the WeatherForecast class from this json). On the right side of the IDE, the Solution Explorer shows the project structure. A purple arrow points from a text box to the `weather.json` file in the `sample-data` folder. The text box contains the text: **クライアントのローカルの json ではなく、サーバー上の json を呼んでいる** (Not calling the local json on the client, but calling the json on the server).

```
1 {  
2   "date": "2018-05-06",  
3   "temperatureC": 1,  
4   "summary": "Freezing"  
5 },  
6 {  
7   "date": "2018-05-07",  
8   "temperatureC": 14,  
9   "summary": "Bracing"  
10 },  
11 {  
12   "date": "2018-05-08",  
13   "temperatureC": -13,  
14   "summary": "Freezing"  
15 },  
16 {  
17   "date": "2018-05-09",  
18   "temperatureC": -16,  
19   "summary": "Balmy"  
20 },  
21 {  
22   "date": "2018-05-10",  
23   "temperatureC": -2,  
24   "summary": "Chilly"  
25 }  
26 }  
27 }
```

この json より
WeatherForecast
クラスを自動生成

クライアントのローカルの
json ではなく、サーバー
上の json を呼んでいる

HTML側

- インスタンス配列から foreach で table の tr タグを作って、表を生成している。

```
16 <table class="table">
17   <thead>
18     <tr>
19       <th>Date</th>
20       <th>Temp. (C)</th>
21       <th>Temp. (F)</th>
22       <th>Summary</th>
23     </tr>
24   </thead>
25   <tbody>
26     @foreach (var forecast in forecasts)
27     {
28       <tr>
29         <td>@forecast.Date.ToShortDateString()</td>
30         <td>@forecast.TemperatureC</td>
31         <td>@forecast.TemperatureF</td>
32         <td>@forecast.Summary</td>
33       </tr>
34     }
35   </tbody>
36 </table>
```

次の機会は。。。

- 日記アプリに進みたいです。

- 連載5回のうちの、二，三回目。
- 「Blazorコンポーネントの開発」

