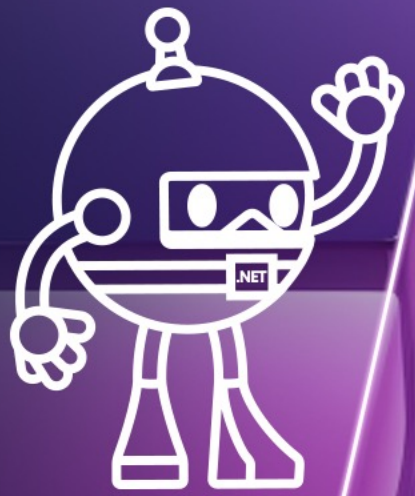
.NET Conf China 2023

2023/12/16
09:30 - 18:00

中国・北京



.NET中文社区

Blazor

中国 · 北京

.NET Conf China 2023

Blazor 全栈渲染模式的原理与应用

The principle and application of Blazor full-stack rendering mode

Microsoft® MVP Most Valuable Professional

杨舜杰 James Yeung

杨舜杰
AntDesign Blazor作者
微软最有价值专家

James Yeung
ElderJames

No need to be adored. No need to wait for a torch.

Edit profile

599 followers · **161** following

@ant-design-blazor
Shanghai
shunjiey@hotmail.com
https://jamesyeung.cn
@shunjiey

Achievements

Beta Send feedback

Pinned

Customize your pins

ant-design-blazor/ant-design-blazor Public
A set of enterprise-class UI components based on Ant Design and Blazor WebAssembly.
C# 5.3k 930

SkyAPM/SkyAPM-dotnet Public
The .NET/.NET Core instrument agent for Apache SkyWalking
C# 1.6k 326

shriek-fx Public archive
An easy-to-use rapid development framework developed on the basis of.NET Core 2.0, following the constraints of domain Driven Design (DDD) specifications, combined with the CQRS architecture to pro...
C# 666 161

vue-router-storage Public
A vue router storage solution. -If your vue application needs to jump to a third party page, and then jump back, want to restore to the original history and continue to operate, the use of this plu...
JavaScript 94 9

NCaptcha Public
Captcha in .NET Core
C# 47 4

CoreProxy Public
Implement a simple Aop using .Net Core library System.Reflection.DispatchProxy
C# 42 15

9 contributions in 2023 in dotnet

Contribution settings

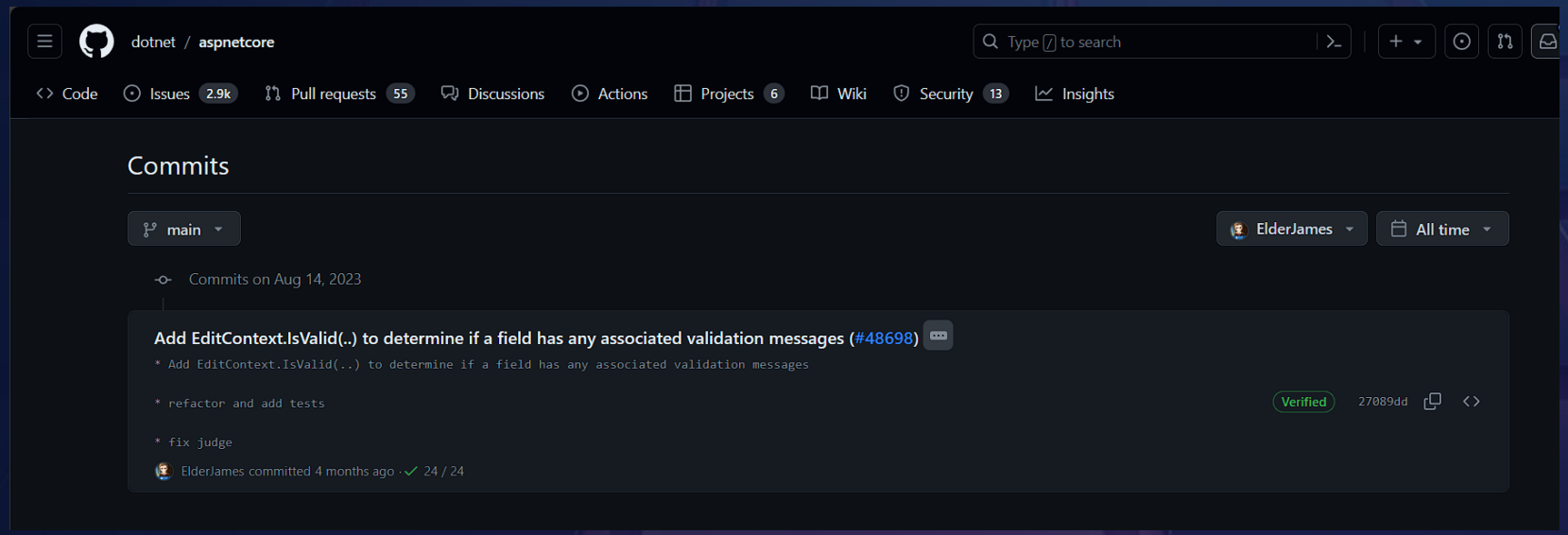
2023 2022 2021 2020 2019 2018 2017 2016 2015 2014

@ant-design-blazor @MLContribution @dotnet More

Activity in **dotnet**

Contributed to aspnetcore

杨舜杰
AntDesign Blazor作者
微软最有价值专家



Blazor

在 .NET 8 中使用 Blazor 进行前端 Web 开发

Frontend web development with Blazor in .NET 8

全栈 Web UI

静态服务端渲染

增强导航&表单处理

流式渲染

增加组件和页面的客户端交互性设置

自动在运行时选择渲染模式

增强

使用组件生成静态 HTML 内容

QuickGrid 表格组件

Sections 插槽组件

路由到特定元素

Circuit 活动监控

强化的身份验证

WebAssembly

Jit 解析器

热重载增强

默认启用 SIMD 和异步处理

内容安全策略兼容

Webcil 打包格式

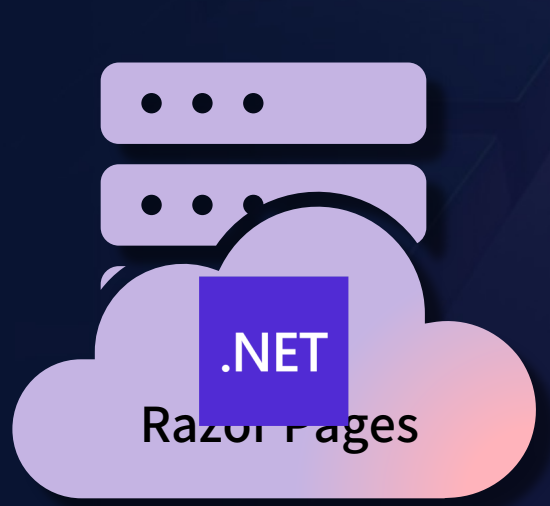
Blazor 全栈渲染模式的原理与应用

The principle and application of Blazor full-stack rendering mode

- 刷新 Blazor 渲染模式认识
- 静态服务端渲染模式的交互性
- 自动交互模式的状态保持
- 现实项目中如何选择渲染模式

Blazor 结合服务端和客户端的优势

The best of server & client with Blazor



Blazor

Server-side rendering (SSR

Client-side rendering (CSR

.NET 8中的 Blazor 与之前对比

	.NET 8之前	.NET 8
重心	交互性	静态和交互平衡，默认静态
交互性模式	同时只能选一种模式	可同时多个，分别设置或自动切换
静态渲染	需切换到 Razor Pages	组件原生支持服务端渲染

Blazor

.NET 8 Blazor 可选的渲染模式

名称	描述	渲染位置	交互
静态服务器	静态服务器端渲染（静态 SSR）	服务器	✗否
交互式服务器	使用 Blazor Server 的交互式服务器端渲染（交互式 SSR）	服务器	✓是
交互式 WebAssembly	使用 Blazor WebAssembly 的客户端渲染(CSR)	客户端	✓是
交互式自动	下载 Blazor 捆绑包后，在后续访问时先使用 Blazor Server 然后使用 CSR 的交互式 SSR	服务器，然后客户端	✓是

PS:
1. Blazor WebApp 默认静态服务端渲染
2. 交互组件默认开启预渲染

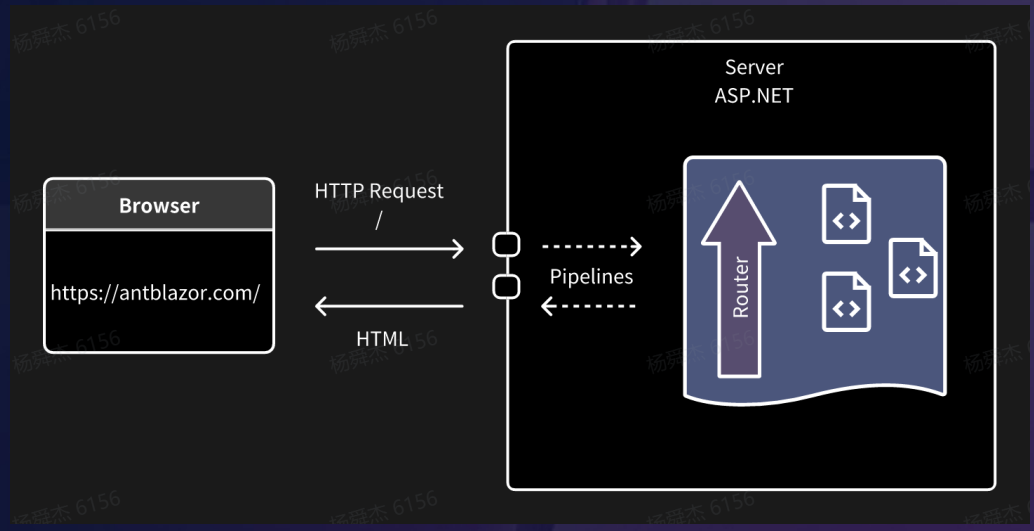
静态服务端渲染原理

服务端接收到用户请求，路由到 Blazor 页面组件

运行页面组件，初始化和渲染逻辑，同时渲染页面中的其他组件

通过页面组件构建的渲染树渲染组件为html

服务端返回html给用户浏览器展示



静态服务端渲染原理

是否倒退到Razor Pages/MVC?

不，是替代并超越

替代：完全覆盖 Razor Pages/MVC 功能，如表单模型绑定

增强1：Blazor 组件模型，可维护性提升

增强2：流式渲染，响应更快

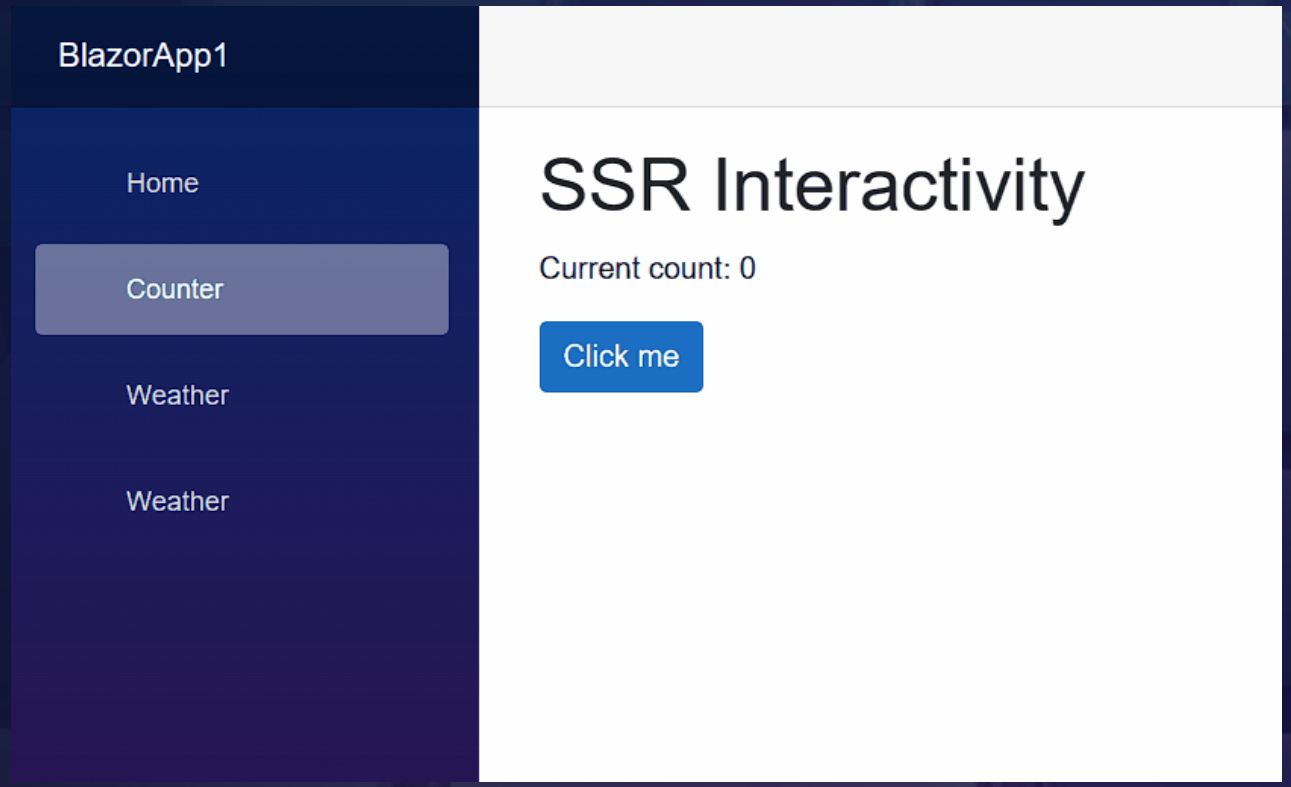
增强3：增强导航/表单，页面跳转更平滑，单页面的体验

Blazor

服务端渲染使 Blazor 成为领先的现代前端框架

静态服务端渲染的交互

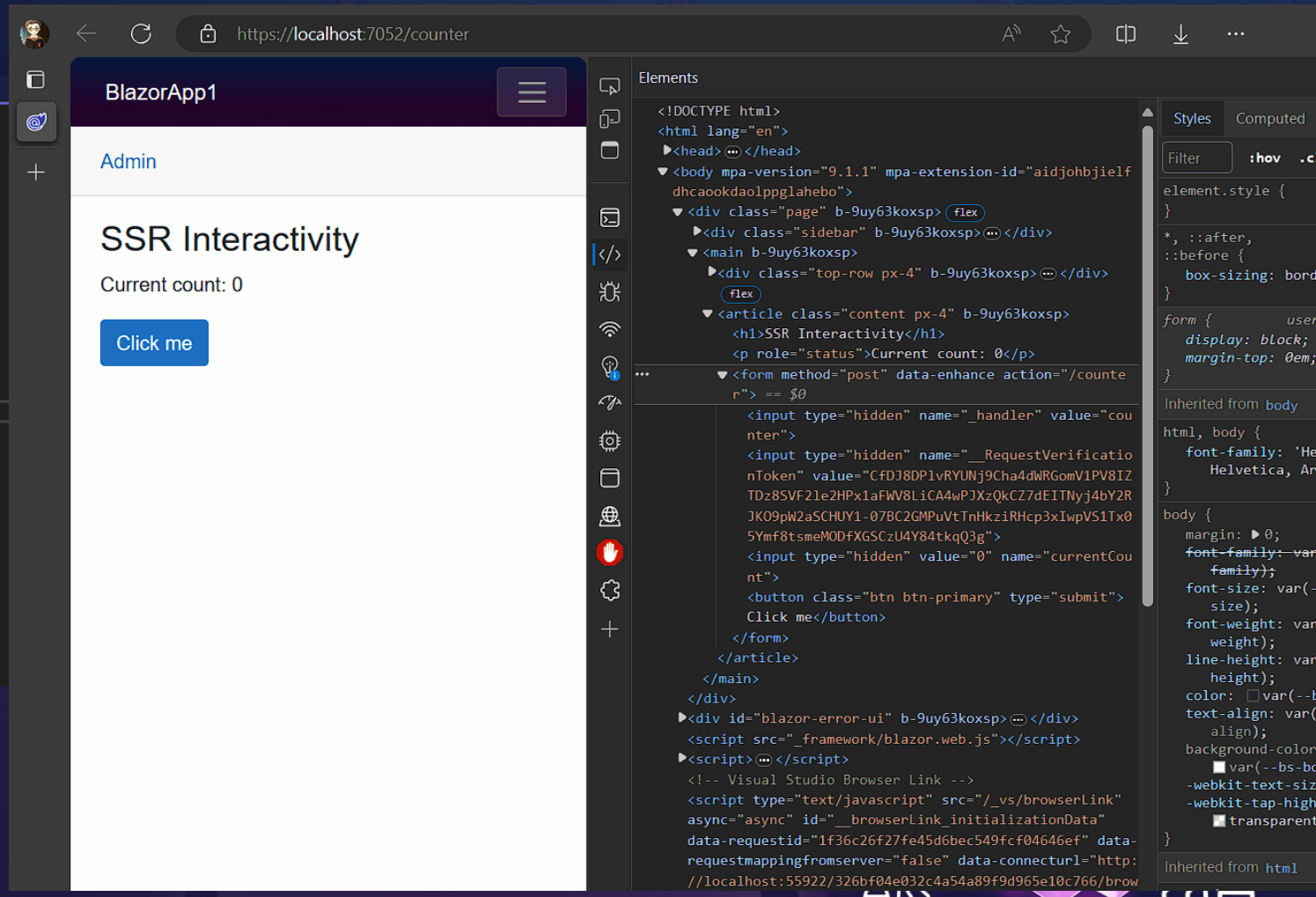
默认：无状态



Blazor

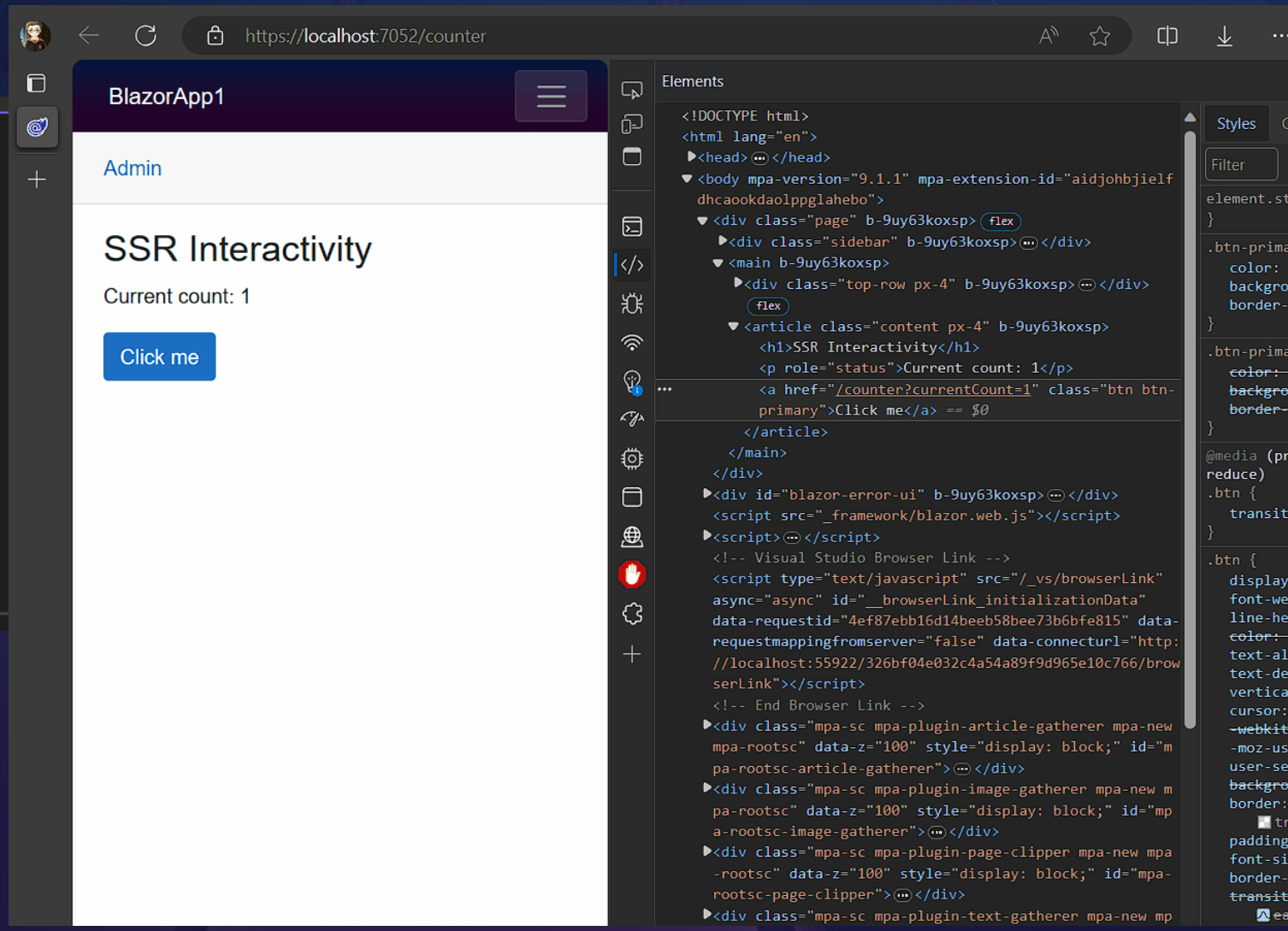
静态服务端渲染的交互—— Form

```razor
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>SSR Interactivity</h1>

<p role="status">Current count: @CurrentCount</p>

<form method="post" data-enhance @onsubmit="IncrementCount" @formname="counter">
    <AntiforgeryToken />
    <input type="hidden" value="@CurrentCount" name="currentCount"/>

    <button class="btn btn-primary" type="submit">Click me</button>
</form>

@code {

    [SupplyParameterFromForm(Name = "currentCount")]
    public int CurrentCount { get; set; }

    private void IncrementCount()
    {
        CurrentCount++;
    }
}
```



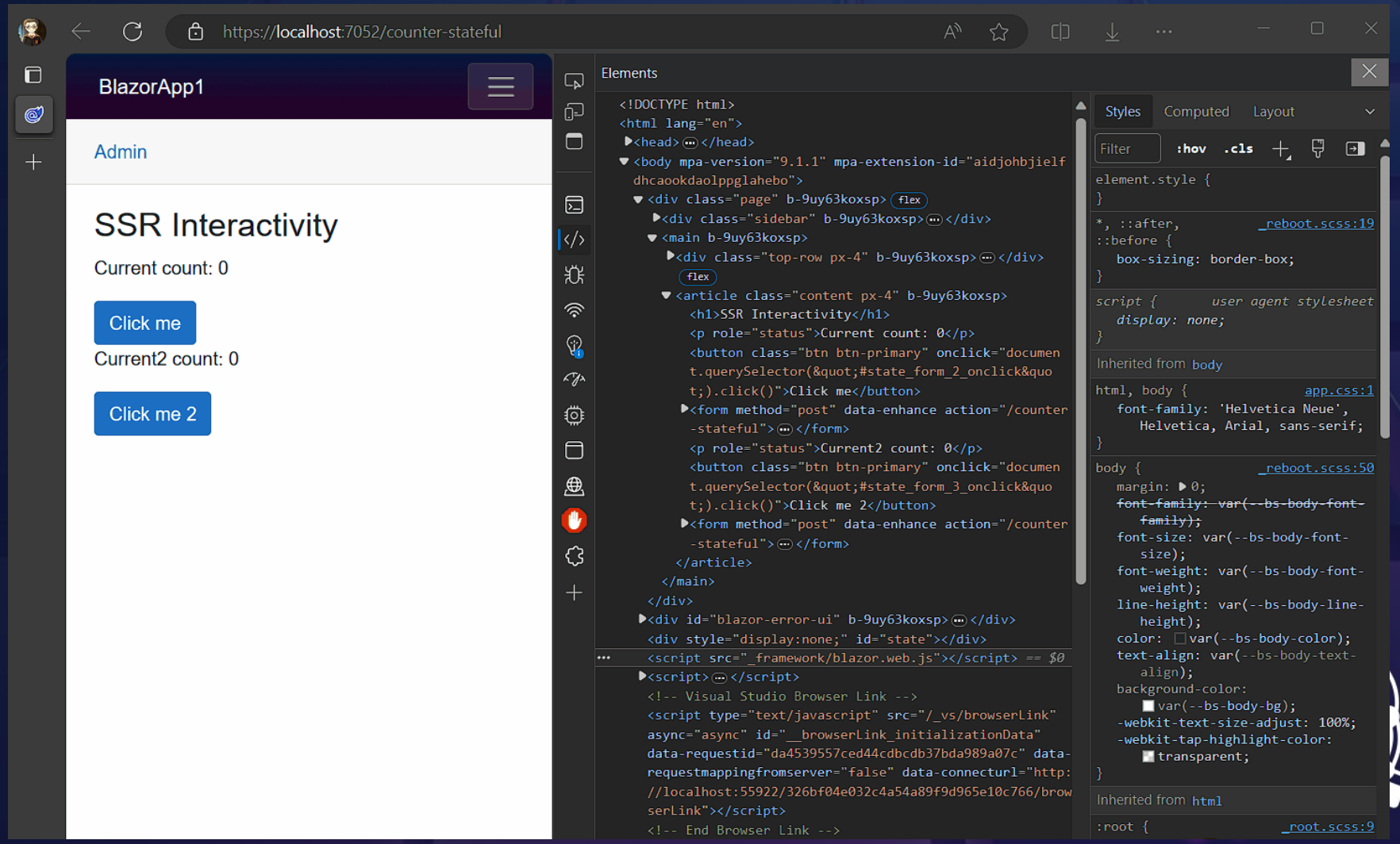
静态服务端渲染的交互——QueryString

```razor
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>SSR Interactivity</h1>

<p role="status">Current count: @CurrentCount</p>


<a href="?currentCount=@CurrentCount" class="btn btn-primary">Click me</a>


@code {

    [SupplyParameterFromQuery(Name = "currentCount")]
    public int CurrentCount { get; set; }

    private void IncrementCount()
    {
        CurrentCount++;
    }
}
```



静态服务端渲染的交互—— 状态组件 (原创)

- 利用 Form 维持状态
- Blazor 组件模型封装
- 自动化收集和恢复状态



静态服务端渲染的交互—— 状态组件 (原创)

- 利用 Form 维持状态
- Blazor 组件模型封装
- 自动化收集和恢复状态

```razor
@page "/counter-stateful"

@inherits StatefulComponentBase

<PageTitle>Counter</PageTitle>

<h1>SSR Interactivity</h1>

<p role="status">Current count: @currentCount</p>

<StatefulElement Tag="button" class="btn btn-primary" @onclick="IncrementCount">Click me</StatefulElement>

<p role="status">Current2 count: @currentCount2</p>

<StatefulElement Tag="button" class="btn btn-primary" @onclick="IncrementCount2">Click me 2</StatefulElement>


@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }

    private int currentCount2 = 0;

    private void IncrementCount2()
    {
        currentCount2++;
    }
}
```

静态服务端渲染的交互局限性

需要同步请求服务端

卡顿不流畅

交互性有限

Blazor

重交互应用，还是需要交互性渲染模式

SSR + Auto

MainLayout.razor

```razor
@inherits LayoutComponentBase
```

1 reference

```
private void ValidateParameters(IReadOnlyDictionary<string, object?> latestParameters)
{
    foreach (var (name, value) in latestParameters)
    {
        // There are many other things we can't serialize too, but give special errors for Delegate because
        // it may be a common mistake to try passing ChildContent when crossing rendermode boundaries.
        if (value is Delegate)
        {
            var valueType = value.GetType();
            if (valueType.IsGenericType && valueType.GetGenericTypeDefinition() == typeof(RenderFragment<>))
            {
                throw new InvalidOperationException($"Cannot pass RenderFragment<T> parameter '{name}' to component '{_componentType.Name}'
            }
            else
            {
                // TODO: Ideally we *should* support RenderFragment (the non-generic version) by prerendering it
                // However it's very nontrivial since it means we have to execute it within the current renderer
                // somehow without actually emitting its result directly, wait for quiescence, and then prerender
                // the output into a separate buffer so we can serialize it in a special way.
                // A prototype implementation is at https://github.com/dotnet/aspnetcore/commit/ed330ff5b143974d9060828a760ad486b1d386ac
                throw new InvalidOperationException($"Cannot pass the parameter '{name}' to component '{_componentType.Name}' with rendermod
            }
        }
    }
}
```

Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.InsertNewFrame(ref DiffContext diffContext, int newFrameIndex)

Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.AppendDiffEntriesForRange(ref DiffContext diffContext, int oldStartIndex, int oldEndIndexExcl, int newStartIndex, int newEndIndexExcl)

Blazor 渲染模式边界与渲染模式传播

- 静态渲染到交互性渲染模式的切换，实际上是不同的 Blazor 运行时在切换
- 通过 @rendermode 设置渲染模式的组件，就是渲染模式边界
- 框架支持组件属性在渲染边界的状态传递
- 无法传递委托、RenderFargment 和 Type 等不能序列化的类型

https://learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0#render-mode-propagation

Blazor 渲染模式边界与渲染模式传播

- 默认渲染模式是静态渲染
- 可以在同级组件中使用不同的交互模式
- 不能在子组件中切换到不同的交互呈现模式。
- 从静态父组件传递给交互式子组件的参数**必须是可JSON序列化的**。

  这意味着不能将 RenderFragment 或子内容从静态父组件传递到交互式子组件。

https://learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0#render-mode-propagation

如何解决子内容在渲染模式边界的传递?

封装成组件，在组件中构建子内容

`WrapperComponent.razor`:

razor Copy

```razor
<SharedMessage>
    Child content
</SharedMessage>
```

`RenderMode10.razor`:

razor Copy

```razor
@page "/render-mode-10"

<WrapperComponent @rendermode="InteractiveServer" />
```

如何解决Layout组件的交互模式?

把路由参数和页面类型传入Layout封装组件，然后重新构造页面组件

```
8 references | James Yeung, 5 days ago | 1 author, 3 changes
public class ReuseTabsRouteData
{
    2 references | James Yeung, 21 days ago | 1 author, 1 change
    public IReadOnlyDictionary<string, object> RouteValues { get; set; }

    2 references | James Yeung, 21 days ago | 1 author, 1 change
    public string PageType { get; set; }

    2 references | James Yeung, 21 days ago | 1 author, 1 change
    public string PageAssembly { get; set; }

    0 references | James Yeung, 20 days ago | 1 author, 2 changes
    public RenderFragment RenderBody()
    {
        return builder =>
        {
            if (RouteData.PageType == null)
                return;

            builder.OpenComponent(0, RouteData.PageType);
            foreach (var routeValue in RouteData.RouteValues)
            {
                builder.AddAttribute(1, routeValue.Key, routeValue.Value);
            }
            builder.CloseComponent();
        };
    }

    1 reference | James Yeung, 5 days ago | 1 author, 2 changes
    public ReuseTabsRouteData(RouteData routeData)
    {
        _routeData = routeData;
        RouteValues = routeData.RouteValues;
        PageType = routeData.PageType.FullName;
        PageAssembly = routeData.PageType.Assembly.FullName;
    }
```

```
@namespace AntDesign.Pro.Template
@inherits LayoutComponentBase

<AntDesign.Pro.Template.BasicLayout RouteData="@RouteData"></AntDesign.Pro.Template.BasicLayout>


@code{
    [CascadingParameter]
    public ReuseTabsRouteData RouteData { get; set; }
}
```

```
</RightContentRender>
<ChildContent>

    @* <ReuseTabs ReuseTabsRouteData="RouteData"></ReuseTabs> *@

    @RouteData.RenderBody()

    @*  @Body *@
</ChildContent>
```

现实项目中如何选择渲染模式

SSR + Auto

核心思想：充分利用Blazor不同渲染模式的优势，根据需求组合使用

前台：静态服务端渲染。如：门户页面、产品落地页、轻交互：
后台：预渲染 + 自动交互式。重交互，业务复杂

Blazor

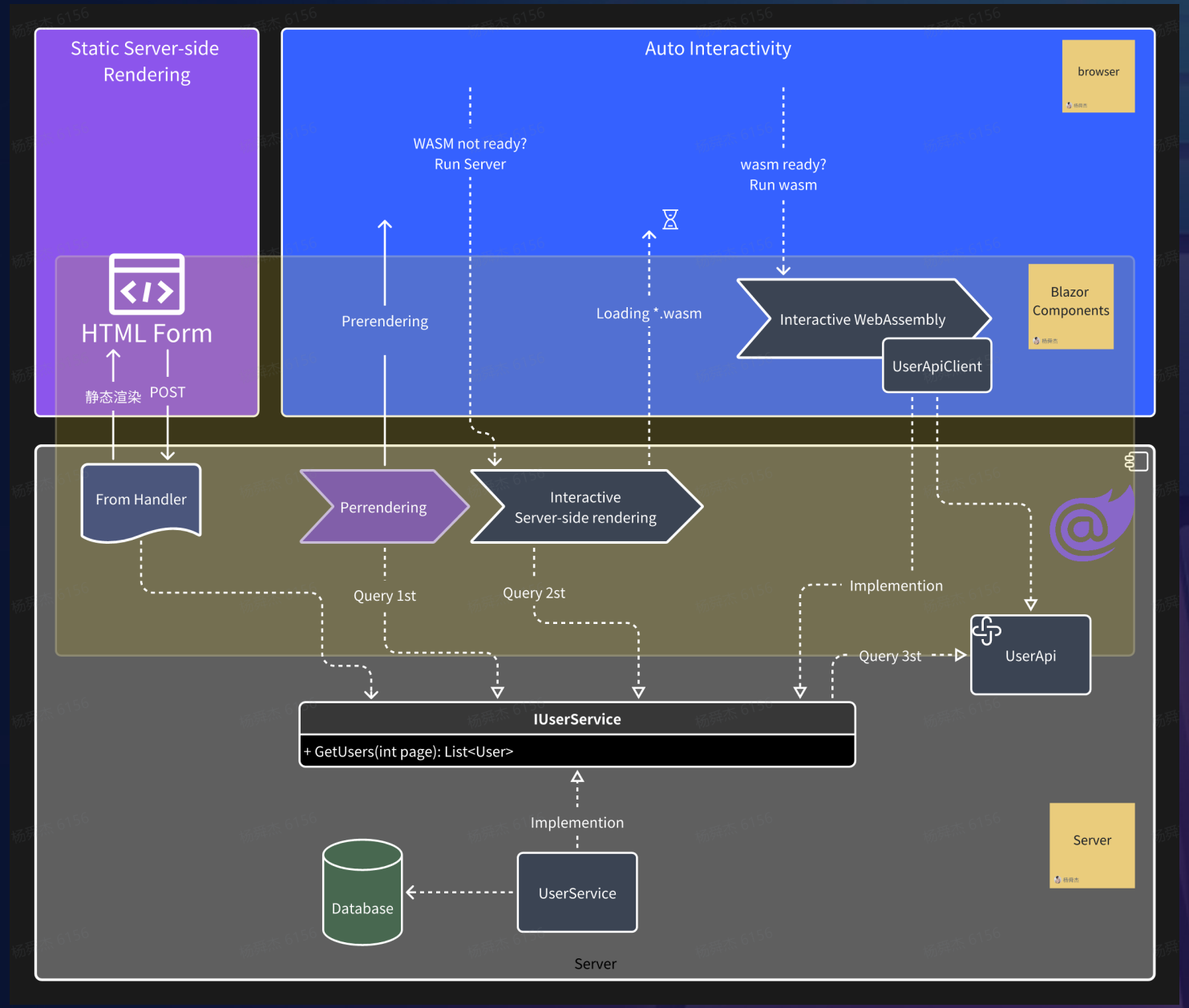
同时兼顾Server和WebAssembly

- 代理模式，类似 RPC
- 代码可用生成

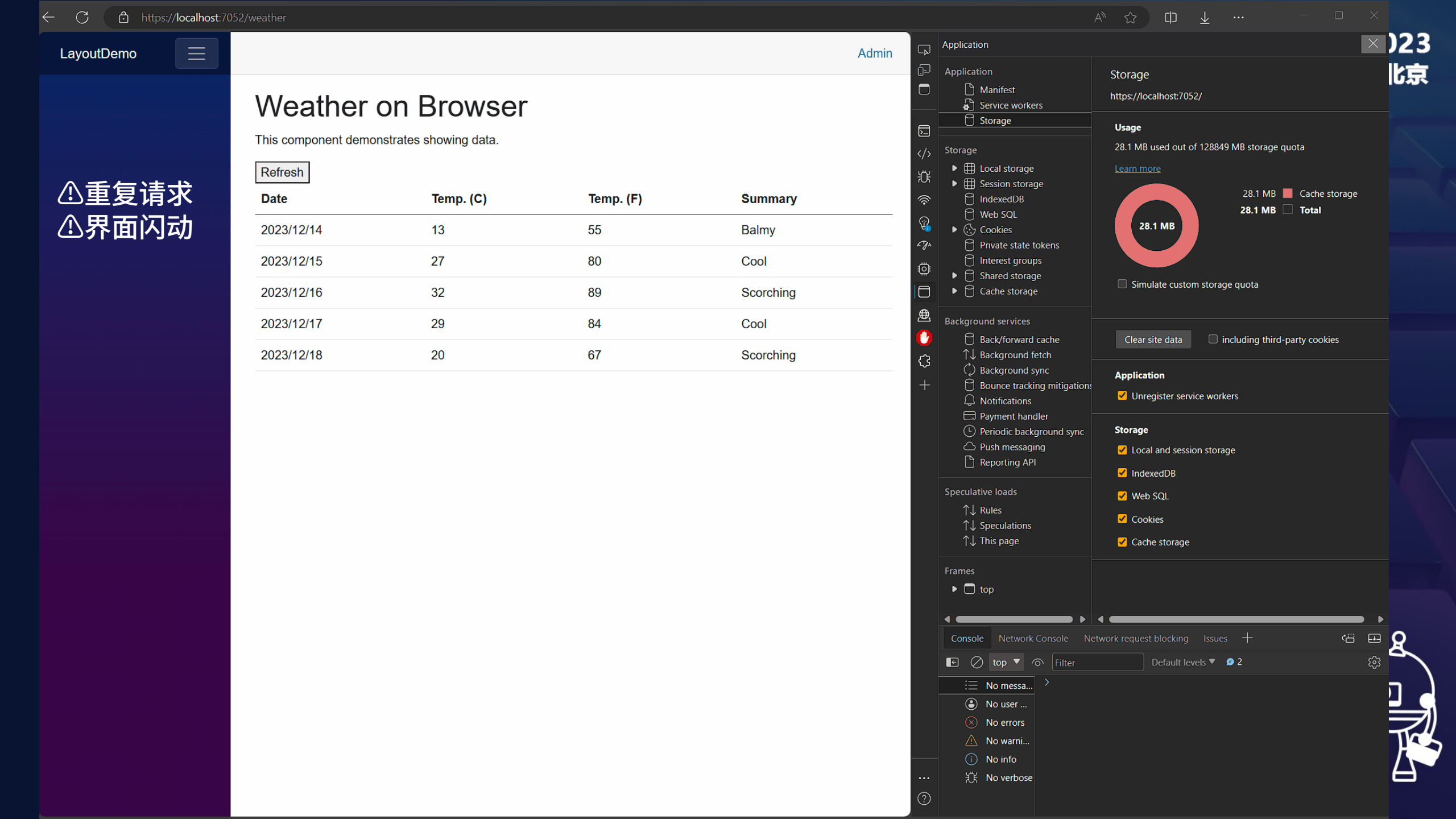
```
9 references | James, 2 days ago | 1 author, 1 change
public interface IWeatherService
{
    7 references | James, 2 days ago | 1 author, 1 change
    public Task<IEnumerable<WeatherForecast>> GetForecasts();
}
```

```
0 references | James, 2 days ago | 1 author, 1 change
public static class WeatherEndpointsExtensions
{
    1 reference | James, 2 days ago | 1 author, 1 change
    public static void MapWeatherEndpoints(this IEndpointRouteBuilder routes)
    {
        routes.MapGet("/api/weather", async ([FromServices] IWeatherService service) =>
        {
            return await service.GetForecasts();
        })
        .WithName("GetAllWeather")
        .Produces<IEnumerable<WeatherForecast>>(StatusCodes.Status200OK);
    }
}
```

```
1 reference | James, 1 day ago | 1 author, 2 changes
public class WeatherService : IWeatherService
{
    4 references | James, 1 day ago | 1 author, 2 changes
    public async Task<IEnumerable<WeatherForecast>> GetForecasts()
    {
        // Simulate asynchronous loading to demonstrate streaming rendering
        await Task.Delay(500);

        var startDate = DateOnly.FromDateTime(DateTime.Now);
        var summaries = new[] { "Freezing", "Bracing", "Chilly", "Cool",
            "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
        };

        var forecasts = Enumerable.Range(1, 5).Select(index => new WeatherForecast
        {
            Date = startDate.AddDays(index),
            TemperatureC = Random.Shared.Next(-20, 55),
            Summary = summaries[Random.Shared.Next(summaries.Length)]
        }).ToArray();

        return forecasts;
    }
}
```


Static Server-side Rendering
Auto Interactivity
browser
WASM not ready?
Run Server
wasm ready?
Run wasm
HTML Form
静态渲染
POST
Prerendering
Loading *.wasm
Interactive WebAssembly
UserApiClient
Blazor Components
From Handler
Perrendering
Interactive Server-side rendering
Query 1st
Query 2st
Implemention
Query 3st
UserApi
IUserService
+ GetUsers(int page): List<User>
Implemention
Server
Database
UserService
Server


.NET
.NET 8
.NET

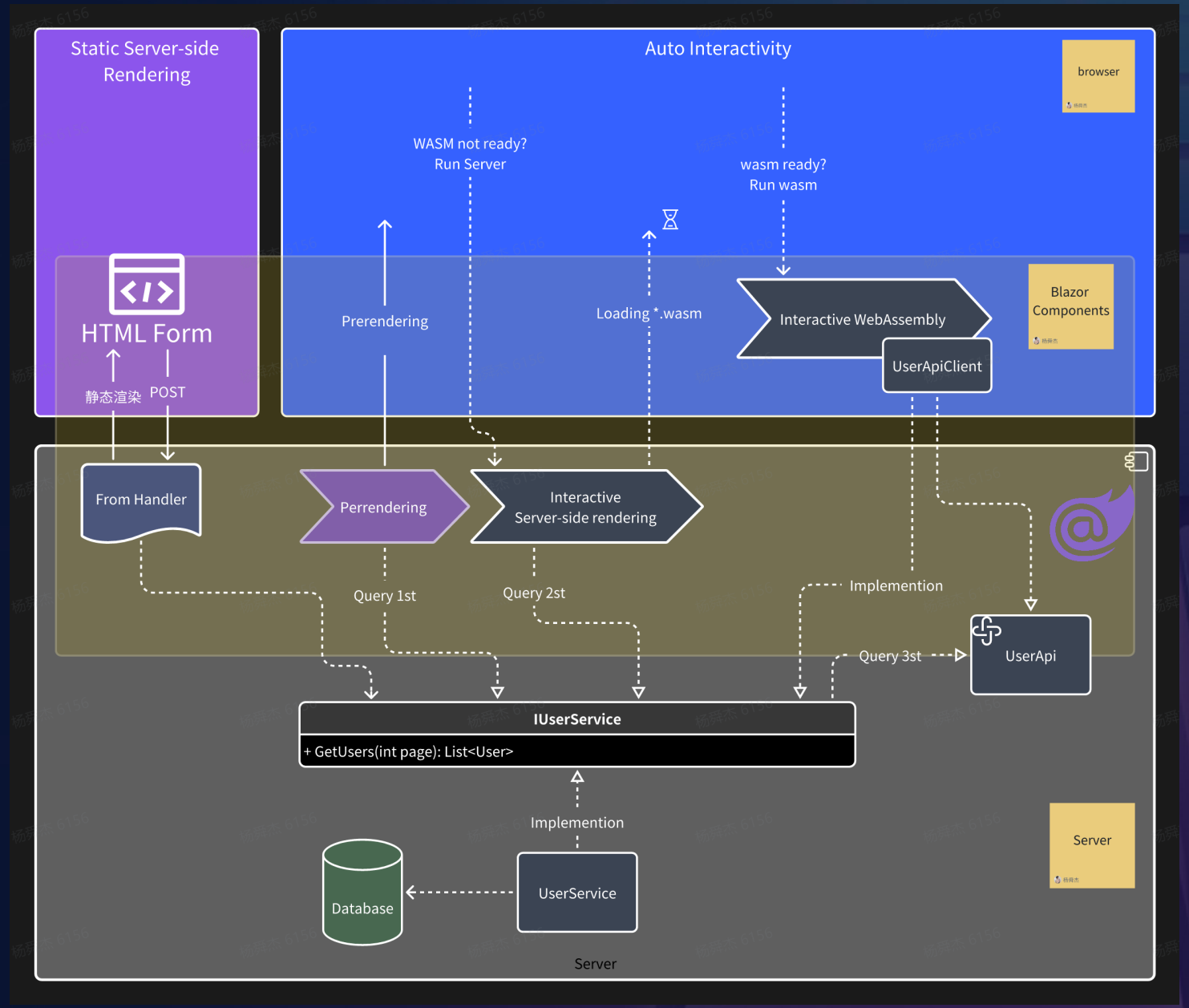

https://localhost:7052/weather
LayoutDemo
Admin
⚠重复请求
⚠界面闪动
Weather on Browser
This component demonstrates showing data.
Refresh
Date
Temp. (C)
Temp. (F)
Summary
2023/12/14
13
55
Balmy
2023/12/15
27
80
Cool
2023/12/16
32
89
Scorching
2023/12/17
29
84
Cool
2023/12/18
20
67
Scorching
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking mitigations
Notifications
Payment handler
Periodic background sync
Push messaging
Reporting API
Speculative loads
Rules
Speculations
This page
Frames
top
Storage
https://localhost:7052/
Usage
28.1 MB used out of 128849 MB storage quota
Learn more
28.1 MB
28.1 MB
Cache storage
28.1 MB
Total
Simulate custom storage quota
Clear site data
including third-party cookies
Application
Unregister service workers
Storage
Local and session storage
IndexedDB
Web SQL
Cookies
Cache storage
Console
Network Console
Network request blocking
Issues
top
Filter
Default levels
2
No messa...
No user ...
No errors
No warni...
No info
No verbose

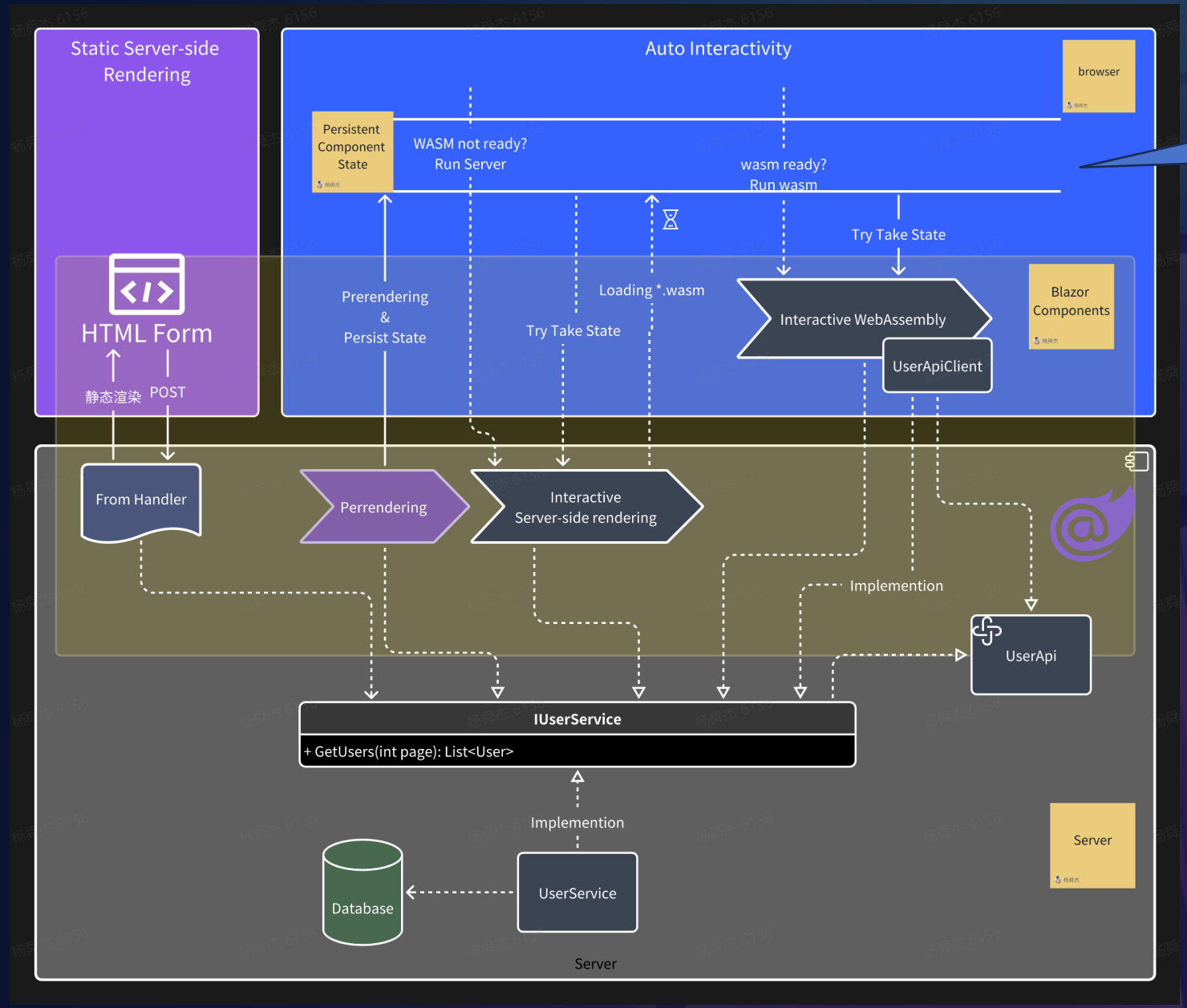
预渲染的优势与局限

能够加快响应速度，是 Blazor 的优势之一。

如果不持久化状态，界面会有闪动、服务端数据重复请求

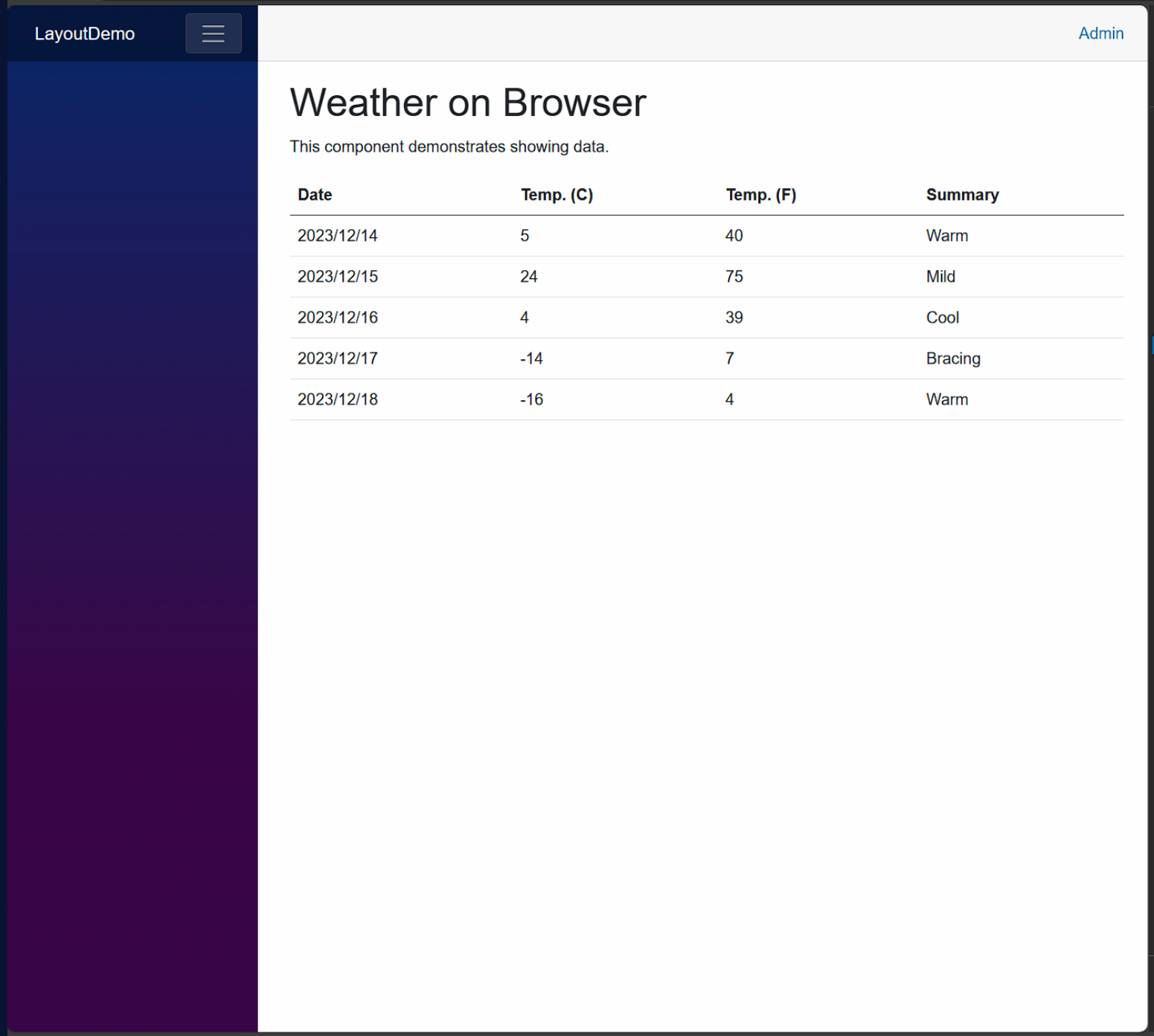
Blazor

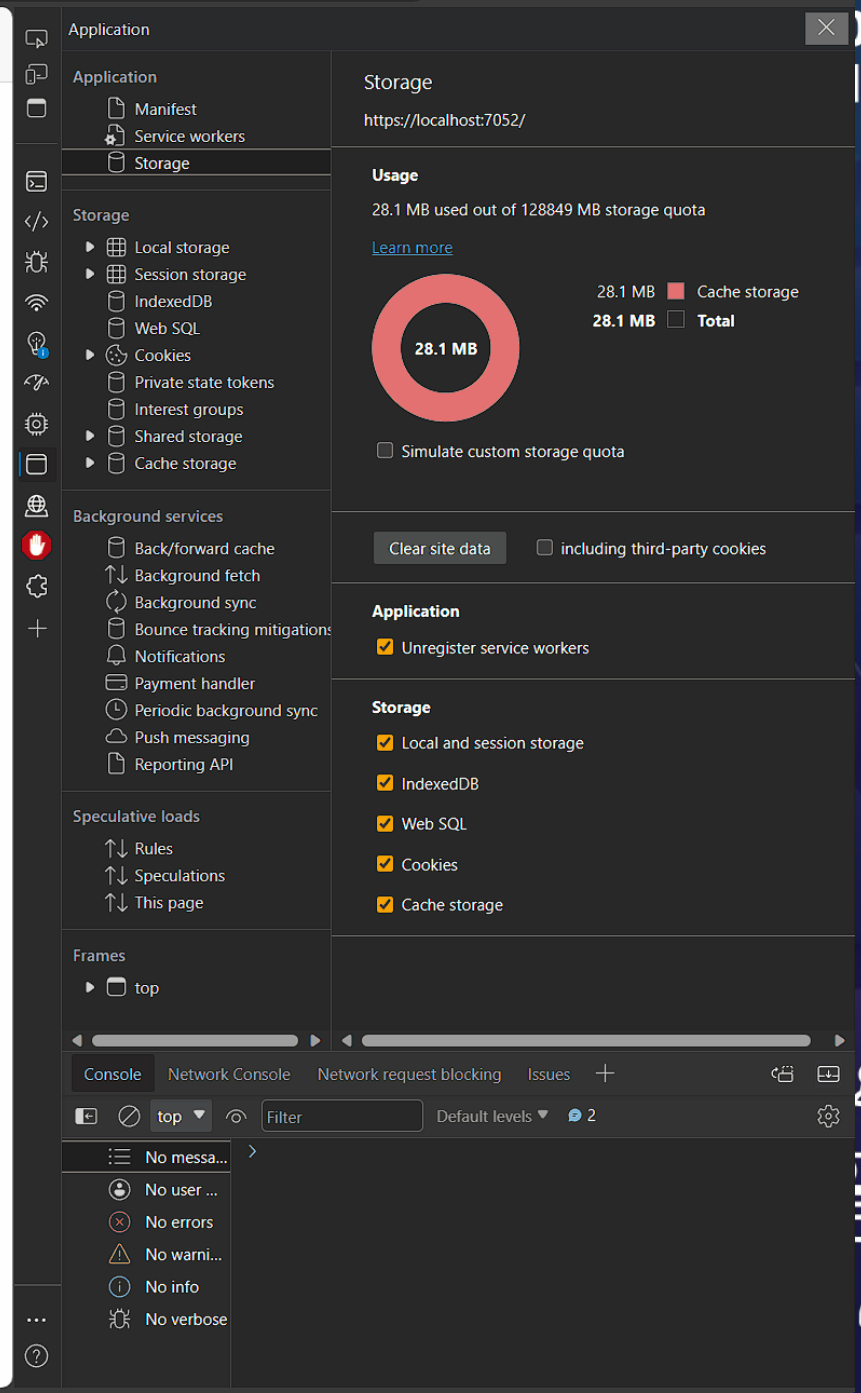

Static Server-side Rendering
Auto Interactivity
browser
WASM not ready?
Run Server
wasm ready?
Run wasm
HTML Form
静态渲染
POST
Prerendering
Loading *.wasm
Interactive WebAssembly
UserApiClient
Blazor Components
From Handler
Perrendering
Interactive Server-side rendering
Query 1st
Query 2st
Implemention
Query 3st
UserApi
IUserService
+ GetUsers(int page): List<User>
Implemention
Server
Database
UserService
Server


Static Server-side Rendering
Auto Interactivity
browser
Persistent Component State
WASM not ready?
Run Server
wasm ready?
Run wasm
Try Take State
Prerendering
&
Persist State
Loading *.wasm
Try Take State
Interactive WebAssembly
UserApiClient
Blazor Components
HTML Form
静态渲染
POST
From Handler
Perrendering
Interactive Server-side rendering
Implemention
UserApi
IUserService
+ GetUsers(int page): List<User>
Implemention
Server
Database
UserService
Server

持久化状态


.NET
.NET 8
.NET


https://localhost:7052/weather
LayoutDemo
Admin
Weather on Browser
This component demonstrates showing data.
Date
Temp. (C)
Temp. (F)
Summary
2023/12/14
5
40
Warm
2023/12/15
24
75
Mild
2023/12/16
4
39
Cool
2023/12/17
-14
7
Bracing
2023/12/18
-16
4
Warm


Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking mitigations
Notifications
Payment handler
Periodic background sync
Push messaging
Reporting API
Speculative loads
Rules
Speculations
This page
Frames
top
Storage
https://localhost:7052/
Usage
28.1 MB used out of 128849 MB storage quota
Learn more
28.1 MB
28.1 MB
Cache storage
28.1 MB
Total
Simulate custom storage quota
Clear site data
including third-party cookies
Application
Unregister service workers
Storage
Local and session storage
IndexedDB
Web SQL
Cookies
Cache storage
Console
Network Console
Network request blocking
Issues
top
Filter
Default levels
2
No messa...
No user ...
No errors
No warni...
No info
No verbose

Blazor

Thank you!

Q&A