Mô hình hoạt động của Blazor WebAssembly

Blazor WebAssembly (thường gọi tắt là Blazor wasm) là mô hình hoạt động thứ hai của Blazor (sau Blazor Server).

Bài viết này sẽ trình bày chi tiết mô hình hoạt động của Blazor wasm giúp bạn phân biệt với mô hình Blazor Server đã trình bày ở bài viết trước. Qua đó bạn có thể xác định xem nó có phù hợp với yêu cầu của mình và có đáng đầu tư thời gian học hay không.

NỘI DUNG CỦA BÀI [Ẩn]

WebAssembly là gì?

Để hiểu được mô hình hoạt động của Blazor wasm, bạn cần hiểu về WebAssembly.

Trong nhiều năm, JavaScript là ngôn ngữ lập trình duy nhất có thể hoạt động thẳng trên trình duyệt (xem như ứng dụng native của trình duyệt). Tất cả các trình duyệt trực tiếp hiểu và thực thi được JavaScript.

WebAssembly (cũng thường gọi tắt là wasm) là một giải pháp tương đối mới để hỗ trợ chạy chương trình trên trình duyệt.

Hiểu một cách đơn giản, WebAssembly là một loại định dạng mã lệnh ở dạng nhị phân kích thước nhỏ gọn mà tất cả các trình duyệt hiện đại đều có thể hiểu và thực thi. Tức là thay vì chỉ hiểu JavaScript như trước, các trình duyệt hiện nay có thể hiểu thêm một "ngôn ngữ" nữa là WebAssembly.

Các trình duyệt thực thi mã WebAssembly ở tốc độ tương tự như đối với ứng dụng JavaScript. Như vậy, chương trình ở mã WebAssembly cũng có thể coi như là một dạng ứng dụng native của trình duyệt.

Khác biệt với JavaScript, WebAssembly được thiết kế làm mã đích cho các ngôn ngữ khác. Nghĩa là nó không được thiết kế để lập trình viên trực tiếp viết mã. Thay vào đó, các ngôn ngữ lập trình thông thường như C/C++ hoặc C# có thể dịch sang mã WebAssembly nếu tạo ra compiler tương ứng cho nó.

Hiện nay tất cả các trình duyệt phổ biến như Chrome, Firefox, Safari, Opera, Edge đều hỗ trợ WebAssembly.

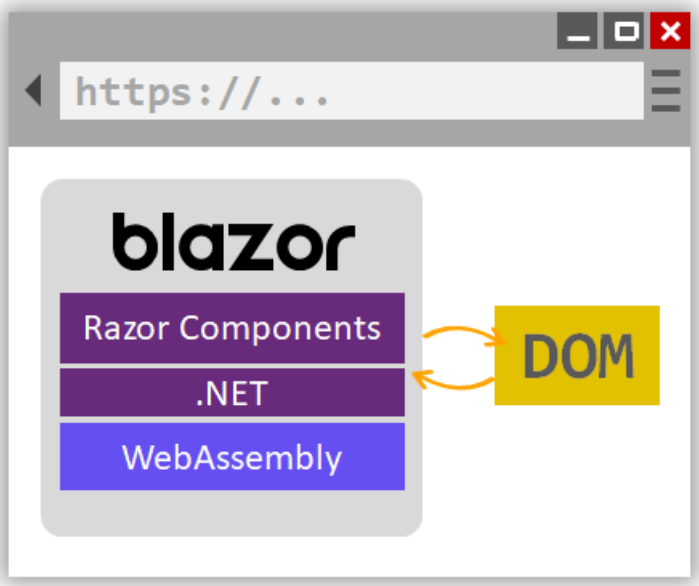
Blazor WebAssembly

Khi đã hiểu thế nào là WebAssembly, có lẽ bạn cũng đoán ra một phần rằng Blazor WebAssembly thực chất chính là C# dịch sang WebAssembly và chạy trên trình duyệt như một ứng dụng native.

Dĩ nhiên, thực tế có khác đôi chút.

Nói một cách chính xác hơn, Microsoft đưa .NET runtime lên WebAssembly (và được gọi là Mono). Code C# vẫn dịch sang .NET bytecode (Intermediate Language) như bình thường. Nhưng giờ đây, bytecode này sẽ được dịch tiếp thành WebAssembly để chạy trên trình duyệt (và trong khuôn khổ quản lý của Mono).

Blazor WebAssembly khi này đóng đúng vai trò là một UI Framework giúp bạn viết chương trình web client chạy trên trình duyệt, tương tự như Angular hay React. Cũng có thể so sánh Blazor WebAssembly với Windows forms hay WPF – các framework giúp bạn phát triển ứng dụng.



Mô hình hoạt động của blazor webassembly

Để đảm bảo hoạt động, các thư viện (dll) cần sử dụng tới trong chương trình Blazor WebAssembly sẽ được tải hết xuống trình duyệt. Trong đó có những thư viện quen thuộc của .NET Core như System.dll.

Tất cả các component bạn viết bằng C# cũng được dịch thành thư viện class và tải xuống trình duyệt. Các component giờ sẽ chuyển sang chạy hoàn toàn trên client, không cần liên quan gì đến server.

Bạn hoàn toàn có thể hình dung một Blazor WebAssembly app giờ chỉ là tập hợp của một số file tĩnh (giống như các file của chương trình desktop) và chạy trong trình duyệt.

Như vậy, mô hình hoạt động của Blazor WebAssembly hoàn toàn khác biệt với Blazor Server như bạn đã biết trong bài viết trước.

Ưu nhược điểm của Blazor WebAssembly?

Với đặc điểm hoạt động như trên, Blazor WebAssembly có thể được sử dụng với vai trò tương đương như Angular. Tất cả thao tác xử lý UI không chạy trên server như đối với Blazor Server nữa. Nó giải phóng server và đẩy công việc về cho trình duyệt, tận dụng khả năng của máy khách cũng như các API của trình duyệt. Tuy nhiên, Blazor WebAssembly cần nhiều thời gian hơn để tải file thư viện cũng như khởi động.

Blazor WebAssembly không cần sử dụng .NET ở server và có thể dùng thậm chí để xây dựng các site tĩnh hoàn toàn chạy trên client (chỉ duy nhất cần tải 1 lần từ server về). Do ứng dụng Blazor WebAssembly được dịch thành một nhóm các file tĩnh, nó thậm chí còn có thể được host trên bất kỳ server nào (vì mục tiêu duy nhất của server chỉ là đẩy các file này về client). Một khi được tải về client, Blazor WebAssembly app có thể chạy hoàn toàn offline bên trong trình duyệt (không cần kết nối Internet).

Blazor WebAssembly trở thành full-stack khi kết hợp với .NET trên server, giúp bạn phát huy hết ưu thế của .NET, tái sử dụng/chia sẻ code, tận dụng tiếp các kỹ năng lập trình .NET sẵn có.

Blazor WebAssembly được tối ưu để render UI. Nó không được thiết kế để chạy những tác vụ quá nặng. Vấn đề nằm ở chỗ, mã trung gian của .NET (mã IL – Intermediate Language) được một trình thông dịch (interpreter) dịch tiếp để thực thi trên WebAssembly. Do vậy, hiệu suất xử lý sẽ bị hạn chế. Nhóm phát triển của Microsoft có kế hoạch hỗ trợ dịch trực tiếp mã nguồn sang mã WebAssembly để tăng hiệu suất.

> **Bonus: Những loại Blazor nào sẽ xuất hiện tiếp?**
> Sau Blazor WebAssembly, nhóm phát triển của Microsoft dự kiến sẽ tiếp tục mở rộng Blazor với Progressive Web App (Blazor PWA), Hybird app (Blazor Hybrid), Native app (Blazor Native).

Kết luận

Mô hình Blazor WebAssembly có những ưu điểm:

- Không phụ thuộc vào .NET server sau khi tải về client.
- Khai thác tài nguyên của client.
- Giảm tải cho server, đồng nghĩa với có thể phục vụ nhiều client hơn.
- Có thể triển khai mà không cần server (ví dụ, từ CDN – Content Delivery Network).

Nhược điểm của Blazor WebAssembly:

- Phụ thuộc vào khả năng của trình duyệt và hiệu suất của client.
- Có yêu cầu cao hơn đối với thiết bị client.
- Kích thước tải về lớn, tốc độ load (lần đầu) chậm hơn.