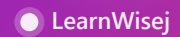


Client-Server-Aufrufe, Serialisierung und Objektzugriff

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Daten und Befehle sicher zwischen .NET-Komponenten und Browser-Widgets austauschen. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.



Fokus: Produktivbetrieb

Daten und Befehle sicher zwischen .NET-Komponenten und Browser-Widgets austauschen.

1. Das Integrationsproblem

Eine brauchbare Integration braucht mehr als das erste Rendering. Der Server muss Client-Methoden aufrufen, der Client muss unter Umständen Server-Methoden aufrufen, und beide Seiten brauchen vorhersehbare Datenverträge.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
Call / CallAsync	Methoden des Client-Widgets aus dem Servercode aufrufen.
DTO	Kleine Data Transfer Objects für stabile serialisierte Verträge verwenden.

3. Lektion aus der Praxis

Wisej.NET stellt eine Kommunikationsschicht bereit, mit der sich Aufrufe vom Server zum Client in C# ganz natürlich anfühlen. Der Server kann eine Client-Methode aufrufen und einfach weiterarbeiten, oder er wartet auf einen Wert aus dem Browser, wenn der Ablauf das erfordert. Umgekehrt kann der Client Server-Methoden aufrufen, die mit `[WebMethod]` bereitgestellt werden – je nach Unterstützung im Callback-Stil oder im Promise/await-Stil. Die Serialisierung ist eine Stärke, aber auch eine Verantwortung. Wisej.NET kann anonyme Objekte, DTOs und primitive Werte serialisieren, doch das heißt nicht, dass jedes Serverobjekt an den Browser geschickt werden sollte. Eine gute Integration verwendet kleine Objekte, die genau auf den Bedarf des JavaScript-Widgets zugeschnitten sind. Sie vermeidet, interne Domänenmodelle, Datenbankentitäten oder sicherheitsrelevanten Zustand in die Client-Optionen durchsickern zu lassen. Benannte Komponenten spiegeln sich im Client-Objektmodell wider. Das ist praktisch für Demos und kontrollierten Anwendungscode, Integrations-Wrapper sollten sich aber nicht von fragilen globalen Pfaden abhängig machen.

Call, CallAsync und EvalAsync

Die drei Aufrufe vom Server zum Client unterscheiden sich in genau einem Punkt: ob der Server wartet. `Call` benennt eine Client-Methode, übergibt serialisierte Argumente und kehrt sofort zurück; der Aufruf wird in die Warteschlange gestellt und mit der nächsten Antwort gesendet. Das ist die richtige Wahl für Befehle, auf die der Server keine Antwort braucht, etwa den Wert der Gauge setzen oder eine Animation zurücksetzen. `CallAsync` benennt ebenfalls eine Client-Methode, liefert aber einen Task, der mit dem Rückgabewert der Methode abgeschlossen wird, sobald der Browser antwortet. `EvalAsync` ist derselbe Mechanismus für einen beliebigen JavaScript-Ausdruck, der in der Seite ausgewertet wird.

Das Warten auf ein Ergebnis vom Client hält den serverseitigen Handler an, bis der Roundtrip zum Browser abgeschlossen ist. Das ist vertretbar, wenn der Ablauf wirklich von dem Wert abhängt – etwa wenn Sie die gerenderte Größe auslesen, bevor Sie ein Layout wählen –, und Verschwendung, wenn der Wert nur protokolliert wird. Die Regel im Lab lautet: standardmäßig `Call` verwenden und nur dann zu einer wartenden Variante greifen, wenn die nächste Anweisung auf dem Server ohne die Antwort nicht ausgeführt werden kann.

- Wird das Ergebnis vom Client im selben Handler verwendet? Falls nicht, verwenden Sie `Call`.
- Was passiert, wenn der Browser nie antwortet, zum Beispiel weil der Tab geschlossen wurde?
- Wird der erwartete Wert validiert, bevor Sie ihm vertrauen?

Serialisierung und camelCase

Jedes Argument, das an den Client übergeben wird, und jeder Wert, der von dort zurückkommt, durchläuft den JSON-Serializer von Wisej.NET. Primitive Werte, Arrays, Dictionaries, anonyme Objekte und gewöhnliche Klassen lassen sich alle serialisieren, und Eigenschaftsnamen werden auf dem Weg nach außen in camelCase umgewandelt: Aus `SelectedIndex` in C# wird in JavaScript `selectedIndex`. Client-Code, der den Namen in PascalCase liest, erhält undefined – ohne jede Fehlermeldung.

Für die Gegenrichtung gelten eigene Regeln. Ein JavaScript-Objekt, das `CallAsync` zurückgibt, kommt auf dem Server als dynamischer Wert an, und der C#-Code muss es über die JavaScript-Namen lesen oder in ein typisiertes DTO umwandeln. Datumswerte, Zahlen außerhalb des sicheren Ganzzahlbereichs von JavaScript und Enums verdienen in jedem Vertrag einen genauen Blick, denn genau diese Werte werden bei der Übertragung am häufigsten verfälscht.

Bei der Serialisierung entstehen auch Datenlecks. Eine Entität mit einer Navigationseigenschaft auf einen Kundendatensatz serialisiert den Kunden gleich mit. Die sichere Gewohnheit lautet: Niemals ein Domänenobjekt über die Leitung schicken, sondern nur ein Objekt, das eigens für diesen Weg gebaut wurde.

Zugriff auf das Client-Objektmodell

Wisej.NET macht seinen Komponentenbaum für Seitenskripte zugänglich. `App.MainPage` ist die aktuelle Hauptseite, benannte Controls sind als Eigenschaften ihres Containers erreichbar, und `widget(id)` liefert jede Komponente anhand ihrer Client-ID. In einer Demo kann ein Skript so bequem eine Schaltfläche greifen, ein Textfeld auslesen oder die Gauge finden, indem es den Baum durchläuft.

Ein Integrations-Wrapper sollte nicht von diesem Baum abhängen. Ein Pfad wie „die Gauge im dritten Panel der Hauptseite“ bricht in dem Moment, in dem ein Entwickler ein Control umbenennt oder in eine

GroupBox verschiebt. Der Wrapper hat bereits eine Referenz auf sich selbst als `this` und auf seinen eigenen Container, und alles, was er vom Server braucht, kommt über Optionen und Aufrufe an. Die Gauge-Befehle im Lab werden alle direkt auf dem Widget aufgerufen und nie über einen Pfad gesucht.

Reservieren Sie das Client-Objektmodell für Skripte auf Seitenebene, die unter der Kontrolle der Anwendung stehen, etwa einen Handler für Tastenkürzel, und werten Sie jede Verwendung innerhalb eines Wrappers als Befund im Code-Review.

DTOs, die klein und stabil bleiben

Ein Data Transfer Object für die Übertragung ist eine Klasse mit einer Handvoll primitiver Eigenschaften, benannt nach dem, was das Widget braucht – und sonst nichts. Das Lab gibt den ausgewählten Client-Zustand der Gauge als ein solches Objekt zurück: den aktuellen Wert, ob er über dem Schwellenwert liegt, sowie die gerenderte Breite und Höhe. Vier Zahlen und ein Boolean, serialisiert in wenigen Dutzend Bytes und in einem Logeintrag gut lesbar.

Kleine DTOs sind stabil, weil sie sich selten ändern. Eine Eigenschaft hinzuzufügen ist unkritisch, denn der Client ignoriert, was er nicht kennt. Eine Eigenschaft umzubenennen oder zu entfernen ist dagegen eine Breaking Change, die versioniert werden muss – und ein DTO mit vier Eigenschaften ist in wenigen Minuten versioniert. Ein DTO, das eine Domänenentität spiegelt, ändert sich bei jeder Änderung der Entität, und jede Änderung ist eine Gelegenheit, den Client unbemerkt zu brechen.

Schreiben Sie die DTOs zuerst, noch vor den Aufrufen, die sie transportieren. Sobald die Strukturen feststehen, die über die Leitung gehen, haben sowohl der Servercode als auch das InitScript etwas Konkretes, gegen das sie getestet werden können.



Gewohnheit für den Produktivbetrieb

Halten Sie klar getrennt, welchen Zustand Wisej.NET besitzt und wie sich das Client-Widget verhält. Dokumentieren Sie den Client-Server-Vertrag, bevor Sie sich im Produktivbetrieb auf die Integration verlassen.



Typischer Fehler

Demo-JavaScript in Masken kopieren, ohne Lebenszyklus, Ressourcen, Event-Verträge, Aufräumen oder Sicherheitsgrenzen zu dokumentieren.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ ``Call``, ``CallAsync`` und ``EvalAsync`` im Servercode nutzen, um mit dem Client-Widget zu interagieren.
- ✓ Die JSON-Serialisierung von Wisej.NET verstehen, einschließlich der camelCase-Schreibung von Eigenschaftsnamen.
- ✓ Das Client-Objektmodell wie ``App.MainPage``, benannte Widgets und ``widget(id)`` angemessen einsetzen.

- ✓ Kleine, stabile DTOs definieren, statt große oder zufällig gewählte Serverobjekte zu übergeben.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab erstellen, der empfohlene Arbeitsablauf, die geforderten Ergebnisse und der Selbstcheck vor der Abgabe.



Praktisches Lab

Ergänzen Sie die wiederverwendbare Gauge um Befehle: Wert setzen, gerenderte Größe auslesen, Animation zurücksetzen und ausgewählten clientseitigen Zustand an den Server zurückgeben.

Empfohlener Arbeitsablauf

- ✓ Öffnen Sie das vorhandene Projekt des Wisej.NET-Integrationskurses und starten Sie es lokal.
- ✓ Ermitteln Sie die Server-Komponente, das Client-Widget, die Ressourcendatei oder den Endpunkt, um die es in diesem Modul geht.
- ✓ Nehmen Sie jeweils nur eine kleine Änderung vor und testen Sie sie im Browser, bevor Sie weitermachen.
- ✓ Halten Sie anbieterspezifisches JavaScript im Adapter oder Wrapper, statt es über mehrere Seiten zu verstreuen.
- ✓ Passen Sie die UI so an, dass Erfolg, Fehler, Fortschritt oder Änderungen in der Kommunikation klar erkennbar sind.
- ✓ Halten Sie mit Screenshots oder Notizen fest, dass die Integration funktioniert, und begründen Sie die Designentscheidung.

Geforderte Ergebnisse

- ✓ Methodenaufrufe vom Server zum Client
- ✓ Ein Beispiel für ``CallAsync`` oder ``EvalAsync``
- ✓ DTO für die Rückgabe des Client-Zustands

Selbstcheck vor der Abgabe

- ✓ Wann sollte Servercode auf ein Ergebnis vom Client warten?
- ✓ Warum spielt die camelCase-Schreibung eine Rolle, wenn Objekte an JavaScript übergeben werden?
- ✓ Warum sollten DTOs klein sein?