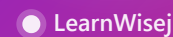


Chiamate client-server, serializzazione e accesso agli oggetti

Guida della lezione · Guida al lab / esame



Guida della lezione

Spostare dati e comandi in sicurezza tra componenti .NET e widget del browser. Che cosa significa per uno sviluppatore che integra con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Spostare dati e comandi in sicurezza tra componenti .NET e widget del browser.

1. Il problema di integrazione

Un'integrazione utile ha bisogno di più del rendering iniziale. Il server deve chiamare metodi del client, il client può aver bisogno di chiamare metodi del server, ed entrambi i lati hanno bisogno di contratti dati prevedibili.

2. Concetti chiave

Concetto	Come usarlo
Call / CallAsync	Invoca metodi del widget client dal codice server.
DTO	Usa piccoli data transfer object per contratti serializzati stabili.

3. La lezione di produzione

Wisej.NET fornisce uno strato di comunicazione che rende naturali da C# le chiamate dal server al client. Il server può chiamare un metodo del client e proseguire, oppure attendere un valore dal browser quando il flusso di lavoro lo richiede. Anche il client può chiamare metodi del server esposti con `[WebMethod]`, usando lo stile a callback oppure lo stile promise/await dove supportato. La serializzazione è un punto di forza ma anche una responsabilità. Wisej.NET sa serializzare oggetti anonimi, DTO e valori primitivi, ma questo non significa che ogni oggetto del server vada inviato al browser. Una buona integrazione usa oggetti piccoli che corrispondono alle esigenze del widget JavaScript. Evita di far trapelare nelle opzioni del client modelli di dominio interni, entità di database o stato sensibile per la sicurezza. I componenti con un nome si riflettono nel modello a oggetti del client. È comodo per le demo e per il codice applicativo sotto controllo, ma i wrapper di integrazione dovrebbero evitare di dipendere da percorsi globali fragili.

Call, CallAsync ed EvalAsync

Le tre chiamate dal server al client differiscono per una sola cosa: se il server aspetta. `Call` indica un metodo del client, passa argomenti serializzati e ritorna subito; l'invocazione viene messa in coda e inviata con la risposta successiva. È la scelta giusta per i comandi di cui il server non ha bisogno di una risposta, come impostare il valore del gauge o reimpostare un'animazione. Anche `CallAsync` indica un metodo del client, ma restituisce un Task che si completa con il valore restituito dal metodo quando il browser risponde. `EvalAsync` è lo stesso meccanismo per un'espressione JavaScript arbitraria valutata nella pagina.

Attendere un risultato dal client sospende il gestore lato server finché il viaggio di andata e ritorno verso il browser non si completa. È accettabile quando il flusso di lavoro dipende davvero dal valore, come leggere la dimensione renderizzata prima di scegliere un layout, ed è uno spreco quando il valore viene solo scritto nel log. La regola del lab è usare `Call` per impostazione predefinita e ricorrere a una variante con attesa solo quando l'istruzione successiva del server non può essere eseguita senza la risposta.

- Il risultato del client viene usato nello stesso gestore? Se no, usa `Call`.
- Che cosa succede se il browser non risponde mai, per esempio perché la scheda è stata chiusa?
- Il valore atteso viene validato prima di fidarsene?

Serializzazione e camelCase

Ogni argomento passato al client e ogni valore restituito da esso passa per il serializzatore JSON di Wisej.NET. Valori primitivi, array, dizionari, oggetti anonimi e classi ordinarie vengono tutti serializzati, e in uscita i nomi delle proprietà vengono convertiti in camelCase, quindi un `SelectedIndex` di C# arriva in JavaScript come `selectedIndex`. Il codice client che legge il nome in PascalCase ottiene undefined e nessun errore.

La direzione opposta ha regole proprie. Un oggetto JavaScript restituito da `CallAsync` arriva sul server come valore dinamico, e il codice C# deve leggerlo con i nomi JavaScript oppure convertirlo in un DTO tipizzato. Date, numeri che superano l'intervallo di interi sicuri di JavaScript ed enum meritano un controllo in ogni contratto, perché sono i valori che più spesso si rovinano durante il transito.

La serializzazione è anche il punto in cui avvengono le fughe di dati. Un'entità con una proprietà di navigazione verso un record cliente serializza anche il cliente. L'abitudine sicura è non passare mai un oggetto di dominio attraverso la comunicazione, ma solo un oggetto costruito apposta per il passaggio.

Accedere al modello a oggetti del client

Wisej.NET espone agli script della pagina il proprio albero dei componenti. `App.MainPage` è la pagina principale corrente, i controlli che hanno un nome sono raggiungibili come proprietà del loro contenitore e `widget(id)` restituisce qualunque componente in base al suo identificatore client. In una demo questo rende facile per uno script prendere un pulsante, leggere una casella di testo o trovare il gauge percorrendo l'albero.

Un wrapper di integrazione non dovrebbe dipendere da quell'albero. Un percorso come «il gauge del terzo pannello della pagina principale» si rompe nel momento in cui uno sviluppatore rinomina un controllo o lo sposta in un group box. Il wrapper ha già un riferimento a sé stesso come `this` e al proprio

contenitore, e tutto ciò che gli serve dal server arriva tramite opzioni e chiamate. I comandi del gauge del lab vengono tutti invocati direttamente sul widget, mai cercati per percorso.

Riserva il modello a oggetti del client agli script a livello di pagina sotto il controllo dell'applicazione, come un gestore di scorciatoie da tastiera, e considera ogni suo uso dentro un wrapper un rilievo da code review.

DTO che restano piccoli e stabili

Un data transfer object per la comunicazione è una classe con una manciata di proprietà primitive, con nomi che riflettono ciò che serve al widget, e nient'altro. Il lab restituisce lo stato selezionato del gauge lato client come un oggetto di questo tipo: il valore corrente, se è sopra la soglia, e la larghezza e l'altezza renderizzate. Quattro numeri e un booleano, serializzati in poche decine di byte e leggibili in una riga di log.

I DTO piccoli sono stabili perché cambiano di rado. Aggiungere una proprietà è sicuro, dato che il client ignora ciò che non conosce. Rinominarne o rimuoverne una è una modifica incompatibile che va versionata, e un DTO con quattro proprietà si versiona in pochi minuti. Un DTO che rispecchia un'entità di dominio cambia ogni volta che cambia l'entità, e ogni modifica è un'occasione per rompere il client senza segnali.

Scrivi prima i DTO, poi le chiamate che li trasportano. Una volta fissate le forme che attraversano la comunicazione, sia il codice server sia l'InitScript hanno qualcosa di concreto su cui essere verificati.



Regola di produzione

Tieni ben distinti la proprietà dello stato in Wisej.NET e il comportamento del widget client. Documenta il contratto client/server prima di affidarti all'integrazione in produzione.



Errore comune da evitare

Copiare nelle schermate il JavaScript delle demo senza documentare ciclo di vita, risorse, contratti degli eventi, pulizia o confini di sicurezza.



Obiettivi di apprendimento

Al termine di questo modulo sarai in grado di:

- ✓ Usare ``Call``, ``CallAsync`` ed ``EvalAsync`` dal codice server per interagire con il widget client.
- ✓ Comprendere la serializzazione JSON di Wisej.NET, compresa la conversione in camelCase dei nomi delle proprietà.
- ✓ Usare in modo appropriato il modello a oggetti del client, come ``App.MainPage``, i widget con un nome e ``widget(id)``.
- ✓ Definire DTO piccoli e stabili invece di passare oggetti server grandi o capitati per caso.

Guida al lab / esame

Che cosa costruirai nel lab pratico, il flusso di lavoro consigliato, i deliverable richiesti e le domande di autovalutazione prima della consegna.



Lab pratico

Aggiungi comandi al gauge riutilizzabile: impostare il valore, leggere la dimensione renderizzata, reimpostare l'animazione e restituire al server una parte selezionata dello stato lato client.

Flusso di lavoro consigliato

- ✓ Apri il progetto esistente del corso di integrazione Wisej.NET ed esegilo in locale.
- ✓ Individua il componente server, il widget client, il file di risorse o l'endpoint coinvolti in questo modulo.
- ✓ Fai una piccola modifica alla volta e provala nel browser prima di andare avanti.
- ✓ Tieni il JavaScript specifico del fornitore dentro l'adapter o il wrapper invece di spargerlo tra le pagine.
- ✓ Aggiorna la UI in modo che si vedano chiaramente successo, errore, avanzamento o cambiamenti nella comunicazione.
- ✓ Raccogli screenshot o note che dimostrano che l'integrazione funziona e spiegano la scelta progettuale.

Deliverable richiesti

- ✓ Chiamate di metodi dal server al client
- ✓ Un esempio di `CallAsync` o `EvalAsync`
- ✓ DTO usato per restituire lo stato del client

Domande di autovalutazione

- ✓ Quando il codice server dovrebbe attendere con `await` un risultato dal client?
- ✓ Perché il `camelCase` è importante quando passi oggetti a JavaScript?
- ✓ Perché i DTO dovrebbero essere piccoli?