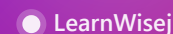


Eventi, gestori e contratti degli eventi dei widget

Guida della lezione · Guida al lab / esame



Guida della lezione

Trasformare le callback del fornitore in eventi .NET stabili senza far trapelare la complessità del fornitore nell'applicazione. Che cosa significa per uno sviluppatore che integra con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Trasformare le callback del fornitore in eventi .NET stabili senza far trapelare la complessità del fornitore nell'applicazione.

1. Il problema di integrazione

I widget di terze parti producono molti eventi nel browser, ma la maggior parte delle schermate applicative ha bisogno solo di pochi eventi significativi lato server, con payload puliti.

2. Concetti chiave

Concetto	Come usarlo
fireWidgetEvent	Inoltra al server gli eventi importanti del widget.
Contratto del payload	Invia al codice .NET dati di evento piccoli e significativi.

3. La lezione di produzione

Integrare gli eventi non è un esercizio meccanico di inoltro. Una griglia può produrre decine di eventi nel browser. Un grafico può generare callback di hover, clic, selezione, zoom, rendering, esportazione e layout. Inviarli tutti al server crea rumore e traffico inutile. Un buon wrapper sceglie gli eventi che rappresentano decisioni di business o cambi di stato posseduto dal server. Per `Wisej.Web.Widget`, `fireWidgetEvent(name, data)` è il percorso diretto verso il `WidgetEvent` lato server. Per i controlli personalizzati, `fireEvent` e `fireDataEvent` si possono gestire tramite `OnWebEvent`, a patto che la configurazione del server colleghi l'evento. La differenza conta perché i controlli personalizzati possono condividere nomi di eventi con gli eventi nativi e devono passare all'implementazione base gli eventi che non gestiscono. Il payload dell'evento va progettato deliberatamente. Invece di inviare l'oggetto evento completo del fornitore, invia `rowKey`, `fieldName`, `value`, `selection` o un altro oggetto compatto su cui il server possa agire direttamente. Il server non dovrebbe mai dover scavare in una struttura specifica del fornitore per trovare l'unico valore che gli interessa, e il payload dovrebbe restare abbastanza piccolo da non diventare un problema di prestazioni se lo si solleva a ogni interazione significativa.

Collegare gli eventi del fornitore al momento giusto

Una callback del fornitore si può collegare solo quando l'istanza del fornitore esiste, quindi il posto naturale è dentro `init`, subito dopo la creazione, mentre `this` si riferisce ancora al widget Wisej.NET. Alcune librerie espongono un'API di eventi sull'istanza, altre accettano callback nell'oggetto options, e alcune richiedono di collegarsi agli elementi DOM che creano. Ogni approccio funziona, purché il gestore catturi il riferimento al widget come insegnato nel Modulo 2.

Le librerie che ricreano la propria istanza quando cambiano certe opzioni hanno bisogno che i gestori vengano ricollegati dopo ogni ricreazione, ed è per questo che il lab tiene il collegamento in un'unica funzione chiamata sia da `init` sia dal percorso di ricreazione in `update`. Collegarli solo in `init` lascerebbe muta la seconda istanza.

Ogni collegamento dovrebbe avere uno scollegamento nel percorso di distruzione del widget. Un gestore del fornitore che sopravvive al suo widget tiene il widget in memoria, e dopo un numero sufficiente di cambi di schermata la perdita diventa visibile come un browser lento.

fireWidgetEvent verso il server

Per un `Wisej.Web.Widget`, il metodo client `fireWidgetEvent(name, data)` è l'unico percorso di cui un'integrazione ha bisogno. Il nome è una breve stringa scelta dall'integrazione, come `thresholdCrossed`, e i dati sono un qualunque oggetto serializzabile. Sul server scatta il `WidgetEvent` del widget con il nome e i dati deserializzati, e il gestore fa uno switch sul nome per decidere che cosa fare.

```
me.gauge.on('change', function (v) {
    if (v > options.threshold && !me.above) {
        me.above = true;
        me.fireWidgetEvent('thresholdCrossed', { value: v });
    }
});
```

L'esempio mostra la disciplina importante: è il client a decidere quando un evento è significativo e a lanciarlo una sola volta, invece di inoltrare ogni modifica e lasciare che sia il server a filtrare. Una classe wrapper derivata da `Wisej.Web.Widget` può gestire internamente `WidgetEvent` e risollevare un evento .NET tipizzato come `ThresholdExceeded`, così il codice dell'applicazione non vede mai i nomi in forma di stringa.

fireEvent e fireDataEvent per i controlli personalizzati

Un controllo personalizzato costruito sul modello di classi client solleva gli eventi in modo diverso.

`fireEvent(name)` solleva un evento senza payload e `fireDataEvent(name, data)` ne solleva uno con payload. Poiché sono normali eventi lato client, il server li riceve solo quando la classe server del controllo ha dichiarato di essere interessata, cosa che il lab descrive come collegare l'evento nella configurazione del server.

Sul server gli eventi arrivano in `OnWebEvent`, che riceve il nome e i dati dell'evento. L'override gestisce i nomi che gli competono e deve passare tutto il resto all'implementazione base, perché un controllo personalizzato condivide la pipeline degli eventi con gli eventi nativi già gestiti dalla sua classe base, come

focus e ridimensionamento. Ingoiare un evento sconosciuto rompe comportamenti che non hanno nulla a che fare con l'integrazione.

La differenza tra i due meccanismi è la domanda di autovalutazione del modulo. `WidgetEvent` è un unico evento generico su un Widget; `OnWebEvent` è il punto di ingresso generale degli eventi di un controllo, condiviso con il framework, e quindi richiede la chiamata alla base.

Payload di cui il codice .NET si può fidare

Il payload di un evento è un contratto, e il lab lo registra in una tabella: nome dell'evento, ogni campo con il suo tipo, quando scatta l'evento e quale evento .NET diventa. Il `pointClicked` del grafico trasporta il nome di una serie, un valore x e un valore y. Il `valueChanged` del knob trasporta solo il nuovo valore. In nessuno dei due payload c'è un oggetto del fornitore, un nodo DOM o un'intera riga di dati.

I payload piccoli restano stabili tra un aggiornamento e l'altro del fornitore, perché sono costruiti da pochi valori estratti dal wrapper e non dalla forma degli argomenti degli eventi del fornitore. Sono anche sicuri: il server tratta ogni campo come non attendibile, lo valida e recupera da sé tutto ciò che è sensibile. Una chiave di riga in un payload è una chiave di ricerca, mai la prova che l'utente possa modificare quella riga.

- Il payload contiene solo primitivi e oggetti piccoli?
- Lo stesso payload potrebbe essere prodotto da una libreria di un altro fornitore?
- Ogni campo viene validato sul server prima di essere usato?



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:

- ✓ Collegare gli eventi del fornitore durante l'inizializzazione o dopo la creazione dell'istanza.
- ✓ Usare `fireWidgetEvent` per inoltrare gli eventi di `Wisej.Web.Widget`.
- ✓ Usare `fireEvent` e `fireDataEvent` con eventi collegati per i controlli personalizzati.
- ✓ Progettare payload degli eventi piccoli, stabili e significativi per il codice applicativo .NET.

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 eventi lato server per il superamento della soglia del gauge, il cambio di valore del knob e il clic su un punto del grafico. Normalizza tutti i payload degli eventi in DTO semplici.

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

- ✓ Tre gestori di eventi lato client
- ✓ Tre gestori di eventi lato server
- ✓ Tabella del contratto dei payload

Domande di autovalutazione

- ✓ Perché non inoltrare ogni evento del fornitore?
- ✓ Qual è la differenza tra ``WidgetEvent`` e ``OnWebEvent``?
- ✓ Che cosa rende stabile il payload di un evento?