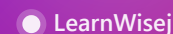


Basi di JavaScript per integrazioni affidabili

Guida della lezione · Guida al lab / esame



Guida della lezione

Padroneggiare i pattern JavaScript che ricorrono di continuo nelle integrazioni di terze parti. Che cosa significa per uno sviluppatore che integra con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Padroneggiare i pattern JavaScript che ricorrono di continuo nelle integrazioni di terze parti.

1. Il problema di integrazione

La maggior parte dei fallimenti di integrazione sono semplici errori JavaScript: contesto perso, elemento contenitore sbagliato, ordine di caricamento degli script, tempistiche asincrone o una chiamata all'API del fornitore fatta prima che il widget esista.

2. Concetti chiave

Concetto	Come usarlo
Contesto	Preserva l'oggetto corretto nelle callback e nel codice asincrono.
Ordine di caricamento	Verifica che script, CSS e librerie del fornitore vengano caricati prima dell'uso.

3. La lezione di produzione

Uno sviluppatore che integra con Wisej.NET ha bisogno di abbastanza dimestichezza con JavaScript da leggere gli esempi del fornitore e tradurli in un adapter. Non serve trasformare uno sviluppatore C# in uno specialista di framework frontend. Serve però gestire con cura contesto, callback, elementi DOM ed esecuzione asincrona. L'errore più comune è dare per scontato che `this` in una callback del fornitore si riferisca ancora al widget Wisej.NET. Le librerie di terze parti raramente preservano il contesto del tuo oggetto. Salva il riferimento in una closure, usa `bind` oppure fai passare la callback da un metodo che ha il contesto corretto. I widget Wisej.NET preservano il contesto nei propri listener, ma le librerie del fornitore sono fuori da quel contratto. Il secondo errore comune è usare l'elemento sbagliato. Alcuni widget vogliono un `div` generico. Altri vogliono un `input`, una `textarea`, un `canvas`, una radice SVG o un elemento specifico identificato da un ID. Il `this.container` di Wisej.NET è il posto sicuro in cui ospitare l'oggetto del fornitore.

Mantenere il this giusto

Dentro le funzioni di un InitScript, `this` è il widget client di Wisej.NET, quindi `this.container` e `this.fireWidgetEvent` sono disponibili. Dentro una callback del fornitore, `this` è quello che ha deciso il

fornitore, di solito la sua stessa istanza o l'elemento DOM che ha sollevato l'evento. Il codice scritto come se il widget fosse ancora nello scope compila, gira e fallisce alla prima callback con un errore di metodo non definito.

```
init: function (options) {  
    var me = this;  
    me.gauge = new VendorGauge(me.container, options);  
    me.gauge.on('change', function (v) {  
        me.fireWidgetEvent('valueChanged', { value: v });  
    });  
}
```

Catturare il widget in una variabile locale prima della callback è la soluzione più semplice e la più facile da leggere in revisione. Chiamare `bind(this)` sulla callback ottiene lo stesso risultato quando la funzione è dichiarata altrove. In entrambi i casi l'istanza del fornitore viene memorizzata sul widget, qui come `me.gauge`, così che `update` e la distruzione possano ritrovarla in seguito. Il codice asincrono segue la stessa regola: un timer, una promise o una callback di fetch vengono eseguiti dopo che la chiamata originale è già tornata, e fino a quel momento sopravvive solo un riferimento catturato.

L'elemento giusto, e solo quello

Ogni libreria del fornitore documenta che cosa si aspetta di ricevere. Alcune accettano qualunque elemento di blocco e costruiscono i propri figli. Altre richiedono un `input` da potenziare, un `canvas` su cui disegnare, una radice `svg` o un elemento con un ID specifico a cui punta il loro CSS. Dare a una libreria di grafici un `div` quando voleva un `canvas` produce un riquadro vuoto e nessun errore, ed è per questo che il lab parte da una prova in HTML puro prima che esista qualunque codice Wisej.NET.

L'host sicuro è sempre `this.container`, l'elemento che il widget Wisej.NET riserva ai contenuti. Se il fornitore ha bisogno di un elemento figlio specifico, crealo dentro il contenitore, per esempio impostando l'inner HTML del contenitore a un `canvas` o a un `input` vuoto e passando poi quel figlio al fornitore. Quello che non deve mai succedere è modificare l'elemento radice del widget, cambiarne le classi o sostituirlo, perché il framework si affida a quell'elemento per layout, visibilità e distruzione.

Un rapido test di revisione: cerca nell'InitScript qualunque elemento diverso da `this.container` passato al fornitore. Ogni altro riferimento a un elemento richiede una giustificazione scritta.

Ordine delle dipendenze per script e stili

Gli script del browser vengono eseguiti nell'ordine in cui sono caricati, e un plugin che estende jQuery lancia un errore se jQuery non è già una variabile globale quando il plugin viene eseguito. Lo stesso vale per una libreria di grafici che si aspetta il proprio core prima dei moduli di estensione, e per un tema CSS che deve essere caricato dopo il foglio di stile base della libreria perché le sue regole prevalgano. In una pagina HTML autonoma lo sviluppatore controlla quest'ordine con la sequenza dei tag script. In Wisej.NET la lista Packages del widget svolge lo stesso ruolo, e le voci vengono caricate nell'ordine in cui sono elencate.

L'ordine del lab è quello più comune: prima jQuery, poi il foglio di stile del fornitore, poi lo script del fornitore e solo alla fine l'InitScript che li usa. Un ordine sbagliato fallisce in modo confuso. L'errore

compare nella console al caricamento, spesso prima che giri qualunque codice del widget, e cita un simbolo non definito della libreria del fornitore invece di qualcosa che ha scritto lo sviluppatore.

- Elenca ogni file che l'esempio del fornitore include, nell'ordine in cui li include.
- Controlla se jQuery o un'altra libreria condivisa è già caricata dall'applicazione, per evitare di caricarne due copie.
- Fissa le versioni nei percorsi dei package, così che un aggiornamento sia una modifica deliberata.

Debug degli script iniettati

Un InitScript viene iniettato nella pagina a runtime, quindi non compare come file nel pannello dei sorgenti del browser e non puoi impostarvi un breakpoint per nome. Aggiungere un commento `sourceURL` alla fine dello script gli dà un nome, e gli strumenti per sviluppatori lo elencano come se fosse un file vero. Un'istruzione `debugger` dentro `init` mette in pausa l'esecuzione quando gli strumenti sono aperti, ed è il modo più rapido per ispezionare `this`, l'oggetto options e il contenitore nel momento in cui viene creato l'oggetto del fornitore.

La console è il secondo strumento. Registrare nel log l'istanza del fornitore dopo la creazione conferma che esiste e mostra quali metodi espone. Ispezionare il widget nel pannello degli elementi conferma che i figli del fornitore sono stati creati dentro il contenitore e non altrove. Il pannello di rete conferma che ogni package è stato caricato con stato 200 e nell'ordine previsto.

Le note di debug del lab registrano dove è memorizzata l'istanza del fornitore e come è stato verificato. Quelle note diventano la prima pagina della guida alla risoluzione dei problemi quando lo stesso widget si comporta male in produzione.



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 closure e bind per preservare il contesto nelle callback asincrone.
- ✓ Individuare l'elemento DOM giusto per un widget ed evitare di modificare in modo scorretto la radice del widget Wisej.NET.
- ✓ Comprendere l'ordine delle dipendenze di script e CSS, in particolare per le librerie basate su jQuery.

- ✓ Usare in modo efficace gli strumenti per sviluppatori del browser, sourceURL, l'ispezione dalla console e le istruzioni debugger.

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

Prendi un esempio in JavaScript puro di un widget gauge o knob e fallo funzionare prima in una pagina HTML isolata. Poi riscrivilo come InitScript di un Widget Wisej.NET.

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

- ✓ Prova funzionante in JavaScript puro
- ✓ InitScript Wisej.NET che preserva il contesto
- ✓ Note di debug che mostrano dove viene memorizzata l'istanza del widget

Domande di autovalutazione

- ✓ Da che cosa ti protegge ``var me = this;``?
- ✓ Quando dovresti usare ``this.container.innerHTML``?
- ✓ Perché una prova in JS puro è utile prima dell'integrazione in Wisej.NET?