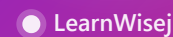


Classi widget riutilizzabili e proprietà adatte al Designer

[Guida della lezione](#) · [Guida al lab / esame](#)

Guida della lezione

Passare da istanze di Widget una tantum a classi lato server riutilizzabili, con proprietà tipizzate e una configurazione controllata. Che cosa significa per uno sviluppatore che integra con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Passare da istanze di Widget una tantum a classi lato server riutilizzabili, con proprietà tipizzate e una configurazione controllata.

1. Il problema di integrazione

Il prototipo funziona, ma lo stesso widget verrà usato su molte schermate. Copiare Packages, Options e InitScript in ogni pagina crea un rischio di manutenzione.

2. Concetti chiave

Concetto	Come usarlo
Proprietà tipizzate	Esponi proprietà .NET stabili invece delle opzioni grezze del fornitore.
Configurazione delle risorse	Tieni la configurazione di package e script dentro la classe wrapper.

3. La lezione di produzione

Una classe widget riutilizzabile è il passo naturale dopo un prototipo riuscito. Lo sviluppatore dell'applicazione non dovrebbe dover ricordare, per ogni schermata, l'ordine degli script del fornitore, il codice di inizializzazione esatto o i nomi grezzi delle opzioni. La classe wrapper centralizza queste decisioni. La classe server deriva ancora da `Wisej.Web.Widget`, ma la superficie pubblica diventa un'API .NET. Per esempio, un gauge può esporre `Value`, `Minimum`, `Maximum`, `Label` e `AnimationEnabled` invece di costringere ogni sviluppatore a modificare un oggetto Options dinamico. La classe può comunque mantenere una via di fuga per le opzioni avanzate, ma il percorso comune dovrebbe essere tipizzato, facile da scoprire e adatto al Designer. È anche qui che contano nomi, valori predefiniti, categorie, descrizioni e type converter. Il wrapper non è più un semplice contenitore di script: sta diventando parte del framework applicativo usato da altri sviluppatori.

Una classe che possiede la propria configurazione

Il lab rifattorizza il prototipo in `IntegrationLab.Controls.SimpleGauge`, una classe che deriva da `Wisej.Web.Widget`. Il suo costruttore, o il suo override di inizializzazione, assegna i Packages, l'InitScript e le Options predefinite che prima venivano scritti su ogni pagina. Una schermata che ha bisogno di un gauge ora dichiara un campo `SimpleGauge`, lo trascina dalla casella degli strumenti e non vede mai una riga di JavaScript.

L'InitScript stesso si sposta in una risorsa incorporata o in una costante della classe, così è versionato insieme al codice C# che ne dipende. I percorsi dei package del fornitore diventano costanti in un unico posto. Quando il fornitore rilascia una nuova versione cambia una sola classe, e ogni schermata la recepisce alla build successiva.

Una classe wrapper è anche il primo punto in cui un registro delle decisioni diventa codice. Il nome della classe, il suo namespace e il commento di riepilogo dicono a che cosa serve il widget e quale libreria del fornitore racchiude, ed è la prima cosa che deve sapere uno sviluppatore fra due anni.

Nascondere le vie di fuga

Quando la classe possiede la configurazione, i membri grezzi diventano pericolosi. Uno sviluppatore che modifica Packages su un'istanza rompe quell'istanza e nessun'altra, il tipo di bug più difficile da trovare. Il wrapper quindi nasconde o protegge Packages, InitScript e l'oggetto Options grezzo, in modo che non compaiano nella finestra Proprietà e non possano essere cambiati con leggerezza dal codice dell'applicazione.

In pratica significa ridefinire i membri come `protected`, oppure marcarli in modo che il Designer non li mostri né li serializzi. La superficie pubblica della classe è allora composta solo dalle proprietà tipizzate e dagli eventi che espone deliberatamente. Gli scenari avanzati si possono comunque servire con un metodo `protected` che permette a una classe derivata di estendere le opzioni, lasciando la via di fuga a chi ne comprende le conseguenze.

La domanda di autovalutazione del lab chiede quali membri andrebbero nascosti agli utenti abituali del wrapper. La risposta è ogni membro il cui uso scorretto potrebbe rompere il widget senza dare segnali.

Proprietà tipizzate che guidano le Options

Una proprietà tipizzata è un piccolo pezzo di C# che legge e scrive nell'oggetto Options presentando un normale tipo .NET. `Value` è un `double` con controllo dell'intervallo, `Minimum` e `Maximum` sono `double` che si validano a vicenda e `Caption` è una stringa. Ogni setter assegna l'opzione in camelCase, e Wisej.NET invia la modifica e chiama `update` sul client esattamente come faceva con il prototipo.

Gli attributi rendono le proprietà adatte al Designer. Un attributo di valore predefinito dice al Designer che cosa non serializzare, una categoria raggruppa le proprietà nella finestra Proprietà e una descrizione spiega ciascuna di esse nel riquadro della guida. Una proprietà costosa da renderizzare, per esempio una che costringe il fornitore a ricostruire il widget, dovrebbe dirlo nella descrizione e assegnare l'opzione solo quando il valore è cambiato davvero.

- Quali modifiche alle proprietà devono forzare un aggiornamento del client e quali sono di sola lettura dopo la creazione.
- Quali valori vengono validati nel setter e quale eccezione solleva un valore non valido.

- Quali valori predefiniti coincidono con quelli del fornitore, così che una nuova istanza venga visualizzata in modo sensato.

Responsabilità del wrapper e responsabilità dell'applicazione

Il wrapper è responsabile del fornitore: caricarne i file, creare e distruggere la sua istanza, tradurre le opzioni e inoltrare gli eventi. L'applicazione è responsabile del significato: quale lettura mostra il gauge, quale soglia conta e che cosa succede quando viene superata. Un wrapper che contiene una chiamata al database o una regola di business ha oltrepassato quel confine, e lo stesso vale per una pagina che contiene codice di inizializzazione del fornitore.

La pagina demo del lab ne è la prova. Mette un `SimpleGauge`, collega `Value` a una lettura proveniente da un servizio e gestisce un evento `.NET`. Non c'è alcun `InitScript` sulla pagina né alcun nome del fornitore nel codice della pagina. Se domani la libreria del fornitore venisse sostituita, la pagina non cambierebbe.

Un utile test del confine è immaginare di distribuire il wrapper come pacchetto NuGet a un altro team. Tutto ciò che dovrebbero modificare dentro il pacchetto appartiene invece all'applicazione.



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:

- ✓ Creare una classe che deriva da `Wisej.Web.Widget`.
- ✓ Nascondere o proteggere Packages, `InitScript` e le Options grezze dove opportuno.
- ✓ Esporre proprietà `.NET` tipizzate che aggiornano in sicurezza l'oggetto Options.
- ✓ Separare la configurazione delle risorse del fornitore dall'uso nell'applicazione.

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

Rifattorizza il prototipo del gauge in `IntegrationLab.Controls.SimpleGauge``, con proprietà tipizzate e registrazione predefinita delle risorse.

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

- ✓ Classe riutilizzabile derivata da Widget
- ✓ Proprietà tipizzate con valori predefiniti
- ✓ Pagina demo che usa la classe senza un InitScript personalizzato sulla pagina

Domande di autovalutazione

- ✓ Quali membri andrebbero nascosti agli utenti abituali del wrapper?
- ✓ Quali modifiche alle proprietà dovrebbero forzare un aggiornamento del client?
- ✓ Che cosa appartiene al wrapper e che cosa all'applicazione?