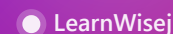


L'architettura di disegno di Wisej.NET

Guida della lezione · Guida al lab / esame



Guida della lezione

Quattro pipeline di rendering e come scegliere fra loro: il Paint dei controlli, che spedisce un'immagine generata sul server; il CellPaint della DataGridView, dentro una singola cella; Wisej.Web.Canvas, che invia comandi di disegno al browser; e System.Drawing.Managed off-screen, che produce byte di immagine — più la scala di decisione che tiene come default i controlli standard e il CSS. Che cosa significa per uno sviluppatore .NET esperto che già costruisce schermate Wisej.NET e ora ha bisogno di grafica personalizzata — e come affrontare questo modulo.



Focus di produzione

Quattro pipeline di rendering e come scegliere tra loro: il Paint di un controllo, che spedisce un'immagine renderizzata dal server, il CellPaint della DataGridView dentro una singola cella, Wisej.Web.Canvas, che invia comandi di disegno al browser, e il System.Drawing.Managed off-screen, che produce byte di immagine — più la scala decisionale che tiene i controlli standard e il CSS come scelta predefinita.

1. Il problema in produzione

VisualOperationsStudio deve mostrare la stessa lettura di una macchina in quattro punti: un quadrante nella pagina delle operazioni, una barra di salute dentro una riga di griglia, un nodo su una mappa di topologia e un PNG allegato al report notturno di turno. Ognuno di questi è "disegnare un numero", eppure affidarsi all'API di disegno che ti ricordi dal desktop produce quattro pezzi di codice con comportamenti radicalmente diversi quanto a CPU, banda e ridimensionamento. Questo modulo mette la scelta della superficie prima del codice di disegno, perché è la superficie a decidere dove vengono prodotti i pixel e che cosa viaggia verso il browser.

2. Concetti chiave

Concetto	Perché è importante
<code>Paint</code> / <code>PaintEventArgs</code>	L'evento di disegno a livello di controllo: il tuo C# gira sul server sul <code>Graphics</code> che l'evento fornisce, dentro il <code>ClipRectangle</code> che riporta, e Wisej.NET trasforma il risultato in un'immagine per il browser.
<code>DataGridView.CellPaint</code>	Lo stesso meccanismo di disegno circoscritto a una sola cella; il suo <code>DataGridViewCellPaintEventArgs</code> deriva da <code>PaintEventArgs</code> e aggiunge <code>RowIndex</code> e <code>ColumnIndex</code> .
<code>UserPaint</code>	L'interruttore di cui una cella o un'intera colonna ha bisogno prima che la griglia sollevi <code>CellPaint</code> , e che mantiene il disegno opt-in per colonna invece che esteso a tutta la griglia.
<code>Wisej.Web.Canvas</code>	Una superficie 2D del browser in modalità immediata che rispecchia il modello <code>CanvasRenderingContext2D</code> di HTML5; le chiamate di disegno vengono eseguite dal client, non renderizzate in un'immagine sul server.
<code>Redraw</code>	L'evento del Canvas in cui l'intera scena viene ricostruita a partire dal tuo modello, perché il canvas conserva i pixel e li perde quando viene ricreato o ridimensionato.
<code>System.Drawing.Managed</code>	Rendering off-screen senza alcun controllo visibile: crea una <code>Bitmap</code> , ottieni il suo <code>Graphics</code> da <code>Graphics.FromImage()</code> e conserva i byte dell'immagine per un report, un export o un download.
<code>Invalidate()</code>	La chiamata che dice a un controllo disegnato a mano che il suo visuale non è più attuale; aggiorna il widget client e la superficie del designer di Visual Studio, quindi va dove lo stato è cambiato, non dentro gli helper di disegno.

3. Il filo della lezione

Quattro pipeline che sembrano tutte normale codice C#

La prima pipeline è l'evento `Paint` del controllo, supportato da ogni controllo Wisej.NET. L'handler riceve un `Wisej.Web.PaintEventArgs` la cui proprietà `Graphics` è una superficie `System.Drawing.Graphics` e il cui `ClipRectangle` identifica il rettangolo che si sta disegnando. Wisej.NET renderizza lo sfondo disegnato e lo invia al browser; la documentazione sul disegno personalizzato descrive il risultato come un PNG che viene trasferito e poi rimosso dalla memoria. È questo a rendere una vecchia routine in stile GDI+ insolitamente riutilizzabile in un'applicazione web.

La seconda pipeline è `DataGridView.CellPaint`, specializzata per le celle. Una cella o un'intera colonna deve avere `UserPaint` impostato a `true`; la griglia solleva allora `CellPaint` con `DataGridViewCellPaintEventArgs`, che porta con sé i consueti `Graphics` e `ClipRectangle` più `RowIndex` e `ColumnIndex`. Si adatta ai visuali compatti — indicatori di stato, minuscoli grafici a barre, sparkline, badge — e alle routine che sarebbero scomode da riprodurre come markup.

La terza pipeline è `Wisej.Web.Canvas`, un canvas del browser in modalità immediata la cui API assomiglia deliberatamente al modello `CanvasRenderingContext2D` di HTML5. La quarta non ha al suo interno alcun controllo: crea una `Bitmap`, ottieni un `Graphics` tramite `Graphics.FromImage()`, disegna e salva o invia l'immagine. Wisej.NET 4 include `System.Drawing.Managed`, che implementa in codice gestito il familiare namespace `System.Drawing` sui target .NET moderni. Nel lab di questo modulo tutte e quattro renderizzano l'unico `TelemetrySample` che la pagina conserva.

Dove gira il codice e che cosa attraversa la rete

Un controllo disegnato dal server non invia al browser la tua chiamata `DrawLine()`. Il server esegue il disegno su una superficie grafica e Wisej.NET trasferisce l'immagine finita — comodo, perché il browser non deve mai capire il tuo algoritmo, e costoso quando la superficie è grande e cambia spesso, dato che CPU del server e byte si pagano a ogni ridisegno. Misura questo costo prima di lasciare che un controllo disegnato a mano si ridisegni su un timer.

Il Canvas ribalta la situazione. L'API `Canvas` di Wisej rappresenta le operazioni del canvas 2D del browser, e quelle operazioni vengono eseguite sul client. `LiveUpdate` è `false` per impostazione predefinita, quindi i comandi emessi durante una richiesta viaggiano con il normale update di Wisej.NET; impostarlo a `true` li invia subito, cosa che va bene per le dimostrazioni incrementali ma non sostituisce l'animazione lato client — un round trip al server per ogni frame è l'architettura sbagliata. Il rendering off-screen non attraversa nulla finché non consegna tu i byte, e `CellPaint` usa lo stesso meccanismo a immagine di `Paint`, limitato a una sola cella.

Modello conservato, pixel immediati

La "modalità immediata" è la parte che spiazzava le persone. Un Canvas ricorda i pixel, non i tuoi oggetti di business, e Wisej.NET non riesegue ogni linea e ogni rettangolo dopo un refresh. Tieni il modello — forme, punti, l'elemento selezionato, zoom, pan, colori — in normali oggetti C#, e rendi `Redraw` deterministico: dati quel modello e la dimensione corrente del canvas, ricostruisce il visuale completo. Nel lab ridimensioni il browser e annoti quali superfici sono tornate da sole.

Gli handler di Paint meritano la stessa disciplina. Trattali come funzioni di rendering: osserva lo stato, disegna, non cambiare nulla. Se la lettura cambia, aggiorna il modello e invalida il controllo; non modificare mai la lettura solo perché è stato richiesto un disegno. Il codice di rendering che muta lo stato è il motivo per cui un visuale finisce per dipendere da quante volte è stato ridisegnato.

Invalidate() al confine di un cambio di stato

`Control.Invalidate()` fa sì che il controllo aggiorni il proprio widget client, e aggiorna anche l'intera superficie nel designer di Visual Studio. In un controllo disegnato a mano è la chiamata naturale quando cambia una proprietà che influisce sul visuale, e la sua sede naturale è il setter della proprietà, protetto in modo che un valore invariato non costi nulla:

```
private double _reading;

public double Reading
{
    get => _reading;
    set
    {
        value = Math.Clamp(value, 0, 100);
        if (_reading == value)
            return;

        _reading = value;
        this.Invalidate();
    }
}
```

Il codice di disegno può ora assumere che il valore sia valido e limitarsi a renderizzare lo stato corrente. Tieni l'invalidazione a quei confini invece di spargerla negli helper di basso livello, dove si trasforma in tempeste di ridisegni che nessuno riesce a tracciare. Un vincolo di progettazione da mettere in conto: la guida al disegno personalizzato è esplicita nel dire che `BackgroundImage` e disegno personalizzato non si possono combinare sullo stesso controllo, perché il disegno prende il posto dello sfondo. Per avere un'immagine dietro il tuo disegno, disegna tu stesso con `Graphics.DrawImage()`.

La scala che tiene il disegno personalizzato per ultimo

La grafica personalizzata deve essere la risposta a un requisito, mai un riflesso ereditato dal lavoro sul desktop. Percorri la scala nell'ordine:

- Può già farlo un controllo standard, un tema, un'icona, un'immagine, una barra di avanzamento, un'estensione per i grafici o il CSS? Preferisci quella strada.
- Il contenuto è naturalmente HTML e leggibile come contenuto del DOM? Valuta il markup prima dei pixel.
- È un piccolo visuale che appartiene allo sfondo di un controllo e cambia a velocità umana? `Paint`.
- È un visuale compatto dentro una cella di griglia? `CellPaint`, dopo averlo confrontato con l'HTML nella cella.
- È una scena interattiva o nativa del browser? `Canvas`. L'output è un file invece di uno schermo? `System.Drawing.Managed` off-screen. È sensibile al frame rate? Tieni il loop sul client.

Due verifiche chiudono la decisione. I pixel non hanno alcun significato per le tecnologie assistive, quindi tutto ciò che è importante ed esiste solo come disegno ha bisogno di un canale testuale parallelo — una label, un tooltip, `AccessibleName` o `AccessibleDescription`. E il trasferimento di rete più la CPU del server fanno parte del budget di disegno, esattamente come le allocazioni fanno parte del budget di un ciclo caldo. Un indicatore disegnato a mano che vince sull'aspetto e perde su entrambi è una decisione che devi essere in grado di difendere in review.



La regola di scegliere prima la superficie

Scegli la superficie di rendering prima di scrivere una sola chiamata di disegno. Chiediti dove vengono prodotti i pixel e che cosa attraversa la rete: l'handler di Paint di un controllo gira sul server e spedisce un'immagine, CellPaint fa lo stesso dentro una singola cella di griglia, il Canvas invia comandi di disegno che il browser esegue e non conserva nulla attraverso un resize, e System.Drawing.Managed produce byte di immagine senza alcun controllo. Tieni il modello semantico in oggetti C#, rendi ogni renderer una funzione deterministica di quel modello, invalida solo quando è cambiata una dipendenza visuale e lascia sempre un equivalente testuale di tutto ciò che è importante ed esiste solo come pixel.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Distinguere il Paint dei controlli, DataGridView.CellPaint, Wisej.Web.Canvas e il rendering off-screen con System.Drawing.Managed.
- ✓ Spiegare il confine server/browser per ciascuna tecnica di disegno.
- ✓ Capire quando un ridisegno invia un'immagine e quando il Canvas invia comandi di disegno al browser.
- ✓ Usare Invalidate() in modo consapevole quando cambia lo stato visuale.
- ✓ Individuare i casi in cui HTML, CSS, un controllo standard di Wisej.NET o un'estensione sono una soluzione migliore del disegno personalizzato.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.

🎯 Lab pratico

Costruisci lo shell di VisualOperationsStudio e rendi un'unica lettura di telemetria su quattro superfici contemporaneamente. Crea una VisualOperationsPage che contiene una sola istanza del modello TelemetrySample, poi presentane il valore come (1) una semplice Label più una ProgressBar, (2) un Panel con un handler di Paint che disegna una barra a partire da PaintEventArgs.Graphics e da e.ClipRectangle, (3) un Wisej.Web.Canvas il cui handler di Redraw ricostruisce la stessa barra dal modello e (4) un Bitmap off-screen generato con Graphics.FromImage e mostrato in una PictureBox. Collega un pulsante che cambia il valore del modello e aggiorna tutte e quattro le superfici tramite Invalidate() e un redraw del Canvas, poi ridimensiona il browser e annota quali superfici sopravvivono e quali devono essere ricostruite.

Punto di partenza

Crea un nuovo progetto Wisej.NET 4.x chiamato VisualOperationsStudio, oppure apri lo shell vuoto se il docente te ne ha fornito uno: una pagina iniziale, un tema di default e nulla di disegnato a mano da nessuna parte. Tutto ciò che c'è in questo lab lo aggiungi tu, e le quattro superfici che costruisci qui sono quelle che ogni modulo successivo trasforma in un indicatore vero, in una griglia disegnata, in un editor di topologia e in un export PNG.

Approccio suggerito

- ✓ Aggiungi una `VisualOperationsPage` e una piccola classe `TelemetrySample` che contiene il nome della macchina, una lettura fra 0 e 100 e un timestamp. Crea esattamente un'istanza come campo della pagina; ogni superficie deve leggere quell'istanza e nessuna può tenerne una copia propria.
- ✓ Disponi sulla pagina quattro aree etichettate. Riempi la prima con una `Label` e una `ProgressBar` agganciate alla lettura, così da avere una baseline senza pixel con cui confrontare le altre tre.
- ✓ Inserisci un `Panel` nella seconda area e gestisci il suo evento `Paint`. Disegna la barra da `e.Graphics`, dimensionala dentro `e.ClipRectangle` e fai il dispose delle pen e dei brush che crei. Leggi la lettura dal modello e non cambiare nulla dentro l'handler.
- ✓ Metti un `Wisej.Web.Canvas` nella terza area e sposta tutto il suo disegno nell'handler di `Redraw`, così che la scena venga ricostruita dal modello invece che accumulata. Nella quarta area genera un `Bitmap` off-screen con `Graphics.FromImage` e mostralo in una `PictureBox`.
- ✓ Aggiungi un pulsante che assegna un nuovo valore al modello e poi aggiorna ogni superficie: aggiorna la label e la progress bar, chiama `Invalidate()` sul pannello disegnato, ridisegna il canvas e ricostruisci l'immagine off-screen dentro `try / catch`. Poi ridimensiona la finestra del browser e scrivi le tue note.

Deliverable richiesti

- ✓ VisualOperationsPage con un unico modello TelemetrySample da cui legge ogni superficie
- ✓ Handler di Paint che disegna da PaintEventArgs.Graphics dentro e.ClipRectangle
- ✓ Handler di Redraw del Canvas che ricostruisce l'intera scena dal modello dopo un ridimensionamento
- ✓ Bitmap off-screen generato con Graphics.FromImage e mostrato nella pagina
- ✓ SurfaceNotes.md che annota che cosa attraversa la rete per ciascuna superficie e quali elementi visivi sopravvivono a un ridimensionamento

Domande di autovalutazione

- ✓ Imposti la lettura del modello dal pulsante ma dimentichi la chiamata a `Invalidate()` sul pannello disegnato. La label e il canvas si aggiornano. Spiega esattamente perché il pannello no, e che cosa sta guardando l'utente finché qualcos'altro non lo costringe a ridisegnarsi.
- ✓ La tua prima versione del Canvas disegna la barra una volta sola al caricamento della pagina invece che da `Redraw`. Descrivi che cosa vede l'utente dopo aver ridimensionato il browser, e perché il `Bitmap` off-screen della quarta area si comporta diversamente.
- ✓ Chi rivede il tuo codice chiede perché la prima area — la semplice `Label` con la `ProgressBar` — sia nel lab. Rispondi elencando i costi che le altre tre superfici pagano e lei no, e indica un requisito che giustificerebbe il pagarli.



La regola di scegliere prima la superficie

Scegli la superficie di rendering prima di scrivere una sola chiamata di disegno. Chiediti dove vengono prodotti i pixel e che cosa attraversa la rete: l'handler di Paint di un controllo gira sul server e spedisce un'immagine, CellPaint fa lo stesso dentro una singola cella di griglia, il Canvas invia comandi di disegno che il browser esegue e non conserva nulla attraverso un resize, e System.Drawing.Managed produce byte di immagine senza alcun controllo. Tieni il modello semantico in oggetti C#, rendi ogni renderer una funzione deterministica di quel modello, invalida solo quando è cambiata una dipendenza visuale e lascia sempre un equivalente testuale di tutto ciò che è importante ed esiste solo come pixel.