

Wisej.Web.Canvas e il modello Canvas 2D di HTML5

Guida della lezione · Guida al lab / esame



Guida della lezione

Leggere la documentazione del Canvas 2D del browser e scrivere l'equivalente Wisej.NET: membri in PascalCase su Wisej.Web.Canvas, angoli in gradi dove il browser usa i radianti, Save e Restore attorno alle trasformazioni, gradienti, pattern, ombre e clipping, e Redraw come unico punto in cui la scena viene ricostruita. 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

Leggere la documentazione Canvas 2D del browser e scriverne l'equivalente in Wisej.NET: membri in PascalCase su Wisej.Web.Canvas, angoli in gradi dove il browser usa i radianti, Save e Restore attorno alle trasformazioni, gradienti, pattern, ombre e clipping, e Redraw come unico punto in cui una scena viene ricostruita.

1. Il problema in produzione

Tutto ciò che VisualOperationsStudio ha disegnato finora è stato prodotto sul server: l'indicatore del Modulo 2 e le celle Health e Trend del Modulo 3 sono resi con `System.Drawing` e arrivano all'utente come immagini. La vista di topologia di cui l'applicazione ha bisogno adesso è fatta di decine di piccoli nodi, archi ed etichette su cui un utente prima o poi farà pan, zoom e selezione: esattamente il lavoro in cui un canvas del browser riesce bene.

`Wisej.Web.Canvas` ti consegna quella superficie da C#, e in cambio ti fa pagare una modalità di guasto diversa. I pixel appartengono al browser, sul server nessuno se li ricorda e un ridimensionamento può cancellarli. Il codice che disegna una volta sola sembra perfetto nel designer e perde la sua scena la prima volta che un utente trascina il bordo di una finestra.

2. Concetti chiave

Concetto	Perché è importante
<code>Wisej.Web.Canvas</code>	Un controllo Wisej che espone a C# la superficie 2D in modalità immediata del browser; le sue chiamate sono comandi di disegno che il browser esegue, non operazioni di <code>System.Drawing.Graphics</code> .
evento <code>Redraw</code>	L'aggancio che la superficie solleva quando le serve di nuovo il suo contenuto; poiché Wisej.NET non conserva i comandi, l'handler deve ricostruire l'intera scena.
corrispondenza in PascalCase	I membri si mappano cambiando le maiuscole — <code>fillRect()</code> diventa <code>FillRect()</code> , <code>strokeStyle</code> diventa <code>StrokeStyle</code> — mentre i valori CSS diventano tipi .NET come <code>Color</code> , <code>Rectangle</code> , <code>Font</code> e <code>Image</code> .
angoli in gradi	Wisej.NET documenta gli argomenti angolari in gradi per semplicità dove il browser usa i radianti, e questo cambia ogni chiamata a <code>Arc()</code> e <code>Rotate()</code> che porti.
<code>Save()</code> / <code>Restore()</code>	Mettono e tolgono dallo stack l'intero stato di rendering — stili, trasformazione, clipping, alpha, impostazioni delle linee e del testo — così che una modifica locale non possa sconfinare nella forma successiva.
<code>TextFont</code> , <code>TextAlign</code> , <code>TextBaseline</code>	Il sostituto Wisej.NET di <code>font</code> , <code>textAlign</code> e <code>textBaseline</code> del browser; il font è un vero <code>System.Drawing.Font</code> .
<code>LiveUpdate</code>	False per default, quindi i comandi viaggiano con il normale aggiornamento della UI della richiesta; true spinge ogni chiamata subito, il che si presta a una rivelazione progressiva e rovina un ciclo di animazione.

3. Il filo della lezione

Leggi la documentazione del browser, poi scrivi il C#

Il Canvas di Wisej.NET è una facciata programmabile dal server sopra il contesto 2D del browser, quindi lo impari traducendo esempi. Gran parte del lavoro è meccanica: `beginPath()` diventa `BeginPath()`, `globalAlpha` diventa `GlobalAlpha`, `setLineDash()` diventa `SetLineDash()`. Dove il browser si aspetta una stringa CSS, Wisej.NET si aspetta un valore .NET: un colore è un `Color`, un font è un `Font` e un'immagine è un `Image` oppure una stringa di origine dell'immagine.

La metà che un riferimento del browser non può darti è che l'elenco documentato dei membri di Wisej.NET non coincide con tutta l'API Canvas moderna. `measureText`, `getImageData`, `roundRect`, `ellipse` e `createConicGradient` non vi compaiono. Tradurre comporta quindi un passo in più: verificare che il membro esista nella versione contro cui compili. Quando non c'è, la risposta onesta è un Widget o del JavaScript lato client, mai un nome di metodo tirato a indovinare.

Redraw è l'unico posto in cui la tua scena esiste

Le chiamate di disegno del Canvas non vengono conservate da Wisej.NET. I comandi vengono emessi, il browser li disegna, e lì finisce il coinvolgimento del server. Quando la superficie viene ridimensionata o

aggiornata il bitmap può essere azzerato, e non esiste alcuna copia in cache da rimettere al suo posto. È questo singolo fatto a separare il codice Canvas che funziona da quello che sembra soltanto funzionare.

Perciò la scena vive in oggetti C#, un solo metodo la ricostruisce per intero e `Redraw` chiama quel metodo. Nel lab `DrawPlayground` si apre con una `ClearRect` su tutta la superficie e riemette ogni comando, ed è per questo che dà lo stesso risultato alla prima visualizzazione, dopo il cambio di un valore e dopo un ridimensionamento.

```
private void canvasPlayground_Redraw(object sender, EventArgs e) => DrawPlayground();

private void DrawPlayground()
{
    canvasPlayground.ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height);

    canvasPlayground.Save();
    canvasPlayground.Translate(240, 160);
    canvasPlayground.Rotate(45);           // browser: ctx.rotate(Math.PI / 4)
    canvasPlayground.FillStyle = Color.Coral;
    canvasPlayground.FillRect(-60, -24, 120, 48);
    canvasPlayground.Restore();
}
```

Due abitudini rompono tutto questo. Disegnare nel `Load` viene eseguito una volta sola, quindi la scena sparisce dopo il primo ridimensionamento. Un handler che ridisegna solo ciò che è cambiato è peggio: funziona mentre sviluppi e lascia una superficie mezza vuota davanti a un utente.

Path, riempimenti, testo, gradienti e ombre

Un path si costruisce prima e si disegna dopo. `MoveTo()` e `LineTo()` aggiungono geometria ma non disegnano nulla; sono `Stroke()` o `Fill()` a disegnare, con gli attuali `StrokeStyle`, `LineWidth`, `LineCap` e `LineJoin`. `BeginPath()` apre un path indipendente, e ometterlo è il classico bug della conversione: un contorno successivo si porta dentro in silenzio la forma precedente. `FillRect()`, `StrokeRect()` e `ClearRect()` agiscono subito, mentre `Rect()` si aggiunge al path che stai costruendo. `CreateLinearGradient()` e `CreateRadialGradient()` restituiscono oggetti che assegni a `FillStyle` o `StrokeStyle`, costruiti a partire da un array di stop di colore, e `CreatePattern()` accetta un `Image` oppure un'origine dell'immagine più un `CanvasRepeatPattern`. Il testo passa da `TextFont`, `TextAlign` e `TextBaseline` prima di `FillText()`; le immagini dagli overload di `DrawImage()`. La profondità arriva dalle proprietà dell'ombra, da `GlobalAlpha` e da `GlobalCompositeOperation` — la composizione è sottile e dipende dal browser, quindi collauda tutto ciò che mandi in produzione usandola.

Save, Restore e lo stack delle trasformazioni

Lo stato di rendering del Canvas è molto più ampio della sola trasformazione, e `Save()` lo mette tutto sullo stack mentre `Restore()` lo rimette a posto. `Translate()`, `Rotate()` e `Scale()` modificano la trasformazione corrente, `Transform()` vi compone sopra un'altra matrice, e `SetTransform()` riporta all'identità e applica quella che fornisci tu.

Racchiudi ogni nodo: save, posiziona, disegna, restore. Senza il restore ogni forma successiva eredita il sistema di coordinate spostato e la scena va alla deriva in un modo che fa impazzire in fase di debug. Un

clipping si comporta allo stesso modo, perché entra a far parte dello stato corrente: un clipping temporaneo è sempre save, path, `Clip()`, disegna, restore. Una sola `Restore()` rimette a posto:

- `FillStyle`, `StrokeStyle` e `GlobalAlpha`
- la matrice di trasformazione corrente e la regione di clipping attiva
- `LineWidth`, `LineCap`, `LineJoin` e il pattern del tratteggio
- le impostazioni del testo e le proprietà dell'ombra

Gradi invece di radianti, e quando `LiveUpdate` si guadagna il suo posto

La documentazione di Wisej.NET sul disegno personalizzato segnala una deviazione voluta: gli argomenti angolari sono espressi in gradi. Un esempio che scrive `ctx.rotate(Math.PI / 4)` diventa `Rotate(45)`, e un cerchio che va da `0` a `Math.PI * 2` diventa da `0` a `360` in `Arc()`. Copiare il valore in radianti compila benissimo e ruota la forma di una frazione di grado, ed è per questo che il lab chiede di scrivere la conversione in un commento accanto alla riga portata.

L'ultima differenza documentata riguarda la consegna. `LiveUpdate` è false per default, quindi i comandi emessi durante una richiesta viaggiano con l'aggiornamento della UI di quella richiesta: un round trip, un ridisegno. True spinge ogni chiamata immediatamente, il che è davvero utile quando una lunga operazione lato server deve mostrare i propri passi, e costoso in tutti gli altri casi. Non è una strada per l'animazione guidata dal server. Tienilo disattivato nel rendering principale, così il costo è qualcosa che hai scelto.



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:

- ✓ Spiegare la corrispondenza fra CanvasRenderingContext2D e Wisej.Web.Canvas.
- ✓ Usare Redraw come aggancio canonico per la ricostruzione della scena.
- ✓ Disegnare path, rettangoli, testo, immagini, gradienti, pattern, ombre e contenuti compositi.
- ✓ Usare Save/Restore e i metodi di trasformazione in sicurezza.
- ✓ Comprendere le differenze documentate come la denominazione C# e gli angoli in gradi.

Guida al lab / esame

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

🎯 Lab pratico

Costruisci in VisualOperationsStudio una pagina playground per il Canvas che porta una serie di esempi Canvas 2D del browser sull'API documentata di Wisej.Web.Canvas. Guida tutto da un metodo DrawPlayground chiamato dall'evento Redraw del Canvas, così che la scena ritorni dopo un ridimensionamento, e aprilo con una ClearRect su tutta la superficie. Disegna un path con contorno, rettangoli pieni, una didascalia collocata, un gradiente lineare e un gradiente radiale, poi racchiudi ogni sezione trasformata fra Save() e Restore() e dimostra Translate, Scale, Rotate e SetTransform. Aggiungi una regione con clipping, una sezione con globalAlpha, una linea tratteggiata e un'ombra, porta un esempio del browser che ruota di $\text{Math.PI} / 4$ nel valore in gradi che Wisej si aspetta, e tieni LiveUpdate disattivato nel rendering principale mostrandolo invece in un piccolo esempio progressivo.

Punto di partenza

Apri VisualOperationsStudio come l'ha lasciato il Modulo 3: il controllo `TelemetryGauge` disegna sul server, `gridOperations` ha le colonne Health e Trend disegnate a mano, e la pagina del Modulo 1 mostra ancora un solo valore di telemetria su quattro superfici. Aggiungi una nuova `CanvasPlaygroundPage` che contiene un solo `Wisej.Web.Canvas` in dock fill, più un piccolo pannello di pulsanti; su quella pagina non c'è ancora nulla che disegni.

Approccio suggerito

- ✓ Scrivi prima `DrawPlayground()` e iscrivilo a `canvasPlayground.Redraw` nel costruttore. Apri il metodo con `ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height)`, disegna una polilinea con contorno usando `BeginPath`, `MoveTo`, `LineTo` e `Stroke`, esegui la pagina e ridimensiona il browser prima di aggiungere qualsiasi altra cosa.
- ✓ Aggiungi la sezione dei riempimenti: due forme `FillRect`, una didascalia collocata con `TextFont`, `TextAlign` e `TextBaseline`, poi una barra riempita con `CreateLinearGradient` e una sfera riempita con `CreateRadialGradient`. Verifica ogni membro contro l'API Canvas di Wisej.NET prima di digitarlo.
- ✓ Costruisci la riga delle trasformazioni come quattro blocchi racchiusi. Ciascuno è `Save()`, poi `Translate`, `Scale`, `Rotate` o `SetTransform`, poi una piccola forma etichettata, poi `Restore()`. Prendi un esempio del browser che chiama `ctx.rotate(Math.PI / 4)`, scrivi `Rotate(45)` e metti la conversione in un commento sulla stessa riga.
- ✓ Aggiungi la sezione dello stato: una regione con clipping costruita come un path seguito da `Clip()` dentro una coppia save/restore, un blocco disegnato con un `GlobalAlpha` ridotto, una riga tratteggiata che usa `SetLineDash` e una scheda disegnata con `ShadowColor`, `ShadowBlur` e gli offset dell'ombra. Commenta di proposito una `Restore()`, guarda il resto della scena spostarsi, poi rimettila.
- ✓ Lascia `LiveUpdate` a false per `DrawPlayground`. Dai a un pulsante un breve ciclo lato server che attiva `LiveUpdate`, disegna un passo di avanzamento per iterazione e lo disattiva di nuovo, poi

metti per iscritto la differenza di traffico e i casi in cui vale la pena accettare quel compromesso.

Deliverable richiesti

- ✓ Playground Canvas la cui scena è disegnata interamente da un metodo DrawPlayground chiamato da Redraw
- ✓ Path, rettangoli, una didascalia collocata, un gradiente lineare e un gradiente radiale usando solo membri documentati di Wisej.Web.Canvas
- ✓ Save() e Restore() attorno a ogni sezione con Translate, Scale, Rotate e SetTransform
- ✓ Clipping, alpha, una linea tratteggiata e un'ombra, più la conversione da radianti a gradi scritta per esteso in un commento
- ✓ Una dimostrazione di LiveUpdate tenuta separata dal rendering principale, con una nota su quando sia appropriato

Domande di autovalutazione

- ✓ Il tuo playground sembra corretto finché un collega non ridimensiona il browser, dopodiché resta visibile solo l'ultima forma che hai disegnato. Che cosa ti dice questo su dove arrivava il resto della scena?
- ✓ Ti serve la larghezza in pixel di una didascalia per dimensionarci un badge. L'API Canvas di Wisej.NET non elenca alcun membro per la misurazione del testo: descrivi due modi per ottenere comunque il layout che vuoi e quanto costa ciascuno.
- ✓ Un revisore ti chiede perché il clipping della tua sezione dello stato sia racchiuso fra `Save()` e `Restore()`, visto che le forme che lo seguono sono disegnate tutt'altrove. Rispondigli.



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.