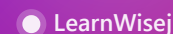


Padroneggiare l'evento Paint dei controlli

Guida della lezione · Guida al lab / esame



Guida della lezione

Controlli disegnati sul server fatti come si deve: `PaintEventArgs.Graphics` e `ClipRectangle`, la geometria separata dal disegno, il `dispose` di ogni pen, brush e font che crei, setter di proprietà che chiamano `Invalidate()` esattamente una volta e il budget di costo del disegno che decide quando un controllo a tema è la risposta migliore. 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

Controlli disegnati sul server fatti come si deve: `PaintEventArgs.Graphics` e `ClipRectangle`, la geometria separata dal disegno, il `dispose` di ogni pen, brush e font che crei, setter di proprietà che chiamano `Invalidate()` esattamente una volta e il budget di costo del disegno che decide quando un controllo a tema è la risposta migliore.

1. Il problema in produzione

Il Modulo 1 ha messo un'unica lettura di telemetria su quattro superfici perché tu potessi vedere dove viene prodotto ogni pixel. Ora `VisualOperationsStudio` vuole quella lettura in sei punti, e copiare un handler `Paint` in sei pannelli è il modo in cui una schermata finisce con sei indicatori leggermente diversi, sei pen di cui nessuno fa mai il `dispose` e un ridisegno ogni volta che cambia un campo che non c'entra nulla. La soluzione è smettere di scrivere handler e scrivere un controllo: un solo `TelemetryGauge` le cui proprietà pubbliche sono l'unica cosa in grado di cambiarne l'immagine, e il cui costo sai esprimere in kilobyte.

2. Concetti chiave

Concetto	Perché è importante
<code>Wisej.Web.PaintEventArgs</code>	L'argomento dell'evento <code>Paint</code> : ti consegna un <code>System.Drawing.Graphics</code> che non hai creato tu e un rettangolo dentro cui disegnare, e il tuo codice gira sul server.
<code>ClipRectangle</code>	La regione che Wisej.NET ha messo a disposizione per questa operazione di disegno; il layout può consultare la client size del controllo, ma il disegno resta circoscritto al rettangolo che ti è stato dato.
helper <code>GaugeGeometry</code>	Una funzione pura che da un <code>Rectangle</code> più un valore normalizzato ricava i limiti dell'anello, gli angoli di apertura, i punti della lancetta e un rettangolo dell'etichetta, così i calcoli si possono verificare senza un controllo vivo.
<code>Invalidate()</code>	L'unica chiamata che chiede a Wisej.NET di renderizzare di nuovo la superficie del controllo; un setter di proprietà la fa esattamente una volta, e solo quando il cambiamento può alterare un pixel.
Proprietà delle risorse	L'handler fa il dispose di ogni <code>Pen</code> , <code>Brush</code> , <code>Font</code> , <code>GraphicsPath</code> e <code>Bitmap</code> che crea, e non fa mai il dispose del <code>Graphics</code> fornito dal framework.
Budget di costo del disegno	Tre numeri per ogni controllo disegnato a mano: la CPU del server spesa a disegnare, la memoria del server per l'immagine temporanea e i byte che il browser deve scaricare e decodificare.
<code>AccessibleDescription</code>	Porta la stessa lettura come testo, così un valore che esiste come arco disegnato resta disponibile anche a un utente che quell'arco non lo vede mai.

3. Il filo della lezione

L'handler, la superficie e il rettangolo che ti è stato dato

Un handler `Paint` di Wisej.NET sembra familiare perché l'oggetto di disegno è davvero `System.Drawing.Graphics` , ma il runtime che lo circonda non è un desktop. Il tuo handler viene eseguito sul server e l'immagine che produce diventa lo sfondo che il browser mostra. Il modello mentale utile è che stai disegnando una bitmap lato server, ed è anche il motivo per cui un ridisegno porta con sé un prezzo di rete.

L'evento fornisce `Wisej.Web.PaintEventArgs` , e due membri contengono tutto ciò che ti serve.

`Graphics` è la superficie attiva: non l'hai allocata tu e non devi rilasciarla. `ClipRectangle` è la regione che Wisej.NET ha messo a disposizione per questa operazione. Ragiona sul layout con il client rectangle del controllo, ma tieni il disegno circoscritto al rettangolo che hai tra le mani.

Per questo le prime due righe di un handler di produzione sono volutamente noiose: verifica che il rettangolo sia abbastanza grande da disegnarci qualcosa, poi passalo all'helper di geometria. Un controllo disposto a larghezza zero, o compresso durante un riflusso responsive, semplicemente esce. Nessun percorso di disegno è tanto importante da giustificare un'eccezione che arriva sullo schermo di un utente.

Forme, path, testo e contenuto trasformato

`Graphics` è un'API di disegno con stato e in modalità immediata, e le famiglie di chiamate che servono all'indicatore sono poche. I contorni arrivano da `DrawArc`, `DrawLine` e `DrawPolygon`; la geometria riempita da `FillPie`, `FillEllipse` e `FillRectangle`; il testo da `DrawString` con un `Font`, un `Brush` e uno `StringFormat`; le forme composte e le curve da un `GraphicsPath` di `System.Drawing.Drawing2D`. Clipping e trasformazioni completano il quadro.

Le trasformazioni sono il punto in cui gli handler di solito sbagliano. Applicare `TranslateTransform` e `RotateTransform` per far ruotare la lancetta è facile; annullarle prima che venga disegnata la didascalia non lo è. Apri un contenitore grafico con `BeginContainer()`, disegna dentro un `try` e metti `EndContainer(state)` nel `finally`: l'implementazione di disegno gestito che Wisej.NET include non ha la coppia `Graphics.Save / Restore`, e il contenitore ne è l'equivalente. La stessa disciplina ricompare nel Modulo 4 con la coppia `Save()` e `Restore()` del Canvas, che invece porta proprio quei nomi.

Il testo merita la stessa cura: `MeasureString` e l'allineamento di `StringFormat` sopravvivono a una didascalia traslata, un offset scritto a mano no. Le immagini hanno una regola in più, perché `Paint` prende il posto dello sfondo. Una filigrana si disegna esplicitamente con `DrawImage`, a partire da un'immagine decodificata una volta sola e posseduta dal controllo, mai da un file aperto dentro l'handler.

Un controllo riutilizzabile le cui proprietà lo invalidano correttamente

Nel lab, `TelemetryGauge` diventa un controllo vero e proprio invece di un handler attaccato a un pannello. Il rendering vive dentro la classe e chi lo usa imposta proprietà di dominio -- `Value`, `Minimum`, `Maximum`, `WarningThreshold`, `CriticalThreshold`. Wisej.NET espone anche un hook protetto `OnWebPaint(Graphics)` su `Control` per gli autori di librerie, mentre l'evento pubblico `Paint` resta il meccanismo di composizione più semplice.

Ogni setter segue un'unica ricetta: limita il valore dentro un intervallo valido, confrontalo con quello memorizzato, esci quando nulla è cambiato, assegna, poi invalida una volta sola. Un valore fuori intervallo viene limitato invece che rifiutato, e un intervallo invertito viene normalizzato invece che lasciato produrre un'apertura negativa. Dopodiché il codice di disegno può assumere che i suoi input siano sensati.

```
public float Value
{
    get => _value;
    set
    {
        var low  = Math.Min(Minimum, Maximum); // un intervallo invertito si
        var high = Math.Max(Minimum, Maximum); // normalizza, non lancia
        var next = Math.Clamp(value, low, high);
        if (Math.Abs(_value - next) < 0.001f)
            return; // non è cambiato nulla di visuale
        _value = next;
        AccessibleDescription = $"{Caption}: {_value:0.0#}";
        Invalidate(); // esattamente una volta per cambiamento reale
    }
}
```

La seconda metà del controllo è `GaugeGeometry` : un helper che riceve un rettangolo e il valore normalizzato e restituisce i limiti dell'anello, un angolo iniziale, un'apertura, i punti della lancetta e il rettangolo dell'etichetta. Non tocca alcun `Graphics` , quindi puoi fare asserzioni sul suo output in un semplice test -- e renderizzare lo stesso indicatore off-screen nel Modulo 6 costa poi quasi nulla.

Chi possiede quale disposable

La proprietà delle risorse è ciò che tiene un controllo disegnato a mano fuori dai guai su un server condiviso. L'handler possiede tutto ciò che crea -- pen, brush solidi e sfumati, font, path, qualsiasi bitmap - e una dichiarazione `using` ne fa il dispose alla fine del metodo. Non possiede `e.Graphics` , e farne il dispose è l'errore classico.

Due abitudini collegate contano altrettanto. Le statiche del framework come `Pens.Red` e `Brushes.Black` sono condivise e vanno lasciate in pace. E un pen mutabile parcheggiato in un campo `static` è peggio di una perdita di memoria: un server Wisej.NET gestisce molte sessioni e può renderizzare per più di una alla volta, quindi lo stato di rendering resta locale a meno che tu non progetti deliberatamente una cache immutabile e thread-safe. Il percorso di disegno inoltre non apre mai una connessione al database, non chiama un servizio e non avvia un timer.

Il budget di costo, e la lettura che nessuno vede

Un controllo disegnato a mano spende in tre punti: CPU del server mentre disegna, memoria del server per l'immagine temporanea e rete più decodifica nel browser per l'immagine inviata. Per il browser una barra di stato e una heatmap a tutta finestra sono entrambe solo immagini, e i loro costi non sono lontanamente paragonabili: misurali tutti e tre quando una superficie cresce. È questo che registra `PaintCost.md` .

La politica di invalidazione è l'altra metà del budget. Scrivi nero su bianco le dipendenze visuali -- di solito valore, didascalia, palette e dimensione -- e ridisegna solo per quelle. Se un valore si muove dieci volte dentro una sola richiesta, aggiorna lo stato dieci volte ma punta a un unico aggiornamento visuale alla fine, e non far mai girare un timer lato server solo per animare un'immagine. Quando i numeri smettono di giustificare i pixel, la risposta onesta è un controllo a tema, un visuale HTML o un'estensione per i grafici.

Infine, un numero disegnato non è un numero accessibile. Un utente che si affida a una tecnologia assistiva riceve un'immagine, quindi l'indicatore pubblica la propria lettura tramite `AccessibleDescription` e la pagina tiene un equivalente testuale accanto alla grafica. Aggiungi un fallback per un font o un'immagine mancanti e uno stato diagnostico minimo al posto di un'eccezione, e il controllo è sicuro da consegnare a un altro team.



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:

- ✓ Gestire l'evento Paint e lavorare con `PaintEventArgs.Graphics` e `ClipRectangle`.
- ✓ Disegnare forme, path, testo, immagini, gradienti e contenuto trasformato con `System.Drawing`.
- ✓ Costruire un controllo disegnato a mano riutilizzabile le cui proprietà pubbliche lo invalidano correttamente.
- ✓ Applicare le regole di durata delle risorse per pen, brush, font, immagini e oggetti graphics.
- ✓ Riconoscere quando il costo del disegno, il trasferimento del PNG o l'accessibilità rendono preferibile un'altra tecnica.

Guida al lab / esame

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

🎯 Lab pratico

Aggiungi a VisualOperationsStudio un controllo TelemetryGauge riutilizzabile che disegna un anello, zone di soglia colorate, una lancetta, un valore numerico e una didascalia. Dagli le proprietà Value, Minimum, Maximum, WarningThreshold e CriticalThreshold, con setter che limitano l'input, escono subito quando nulla è cambiato e chiamano Invalidate() una volta sola. Metti i calcoli delle coordinate in un helper puro GaugeGeometry che riceve un Rectangle e restituisce i limiti dell'anello, gli angoli di apertura, i punti della lancetta e il rettangolo dell'etichetta, poi disegna a partire da quei valori con pen, brush e un GraphicsPath racchiusi in using, lasciando in pace il Graphics che ti viene passato. Gestisci senza eccezioni un controllo a dimensione zero, un intervallo invertito e un valore fuori intervallo, ed esponi la lettura tramite AccessibleDescription così da non affidarla ai soli pixel.

Punto di partenza

Apri VisualOperationsStudio dal Modulo 1, dove VisualOperationsPage contiene già un unico modello TelemetrySample e ne mostra il valore su quattro superfici, incluso un Panel con un handler Paint inline che disegna una barra. In questo lab quell'handler inline viene sostituito: aggiungi al progetto una classe TelemetryGauge, metti tre istanze sulla pagina e lega ciascuna a una lettura dello stesso modello.

Approccio suggerito

- ✓ Aggiungi una classe TelemetryGauge che deriva da un controllo Wisej.NET, dotata di campi di appoggio più Value, Minimum, Maximum, WarningThreshold, CriticalThreshold e Caption, e scrivi un setter che limita, confronta, esce subito e chiama Invalidate() una volta sola -- poi replica quella forma su tutti gli altri.
- ✓ Scrivi GaugeGeometry come helper puro: un metodo che riceve il Rectangle disegnabile e il valore normalizzato e restituisce i limiti dell'anello, l'angolo iniziale e quello di apertura, i punti della lancetta e il rettangolo dell'etichetta, senza alcun riferimento a Graphics in tutto il file.
- ✓ Aggancia l'handler di Paint (oppure esegui l'override dell'hook di disegno) e disegna rigorosamente a partire dal risultato della geometria: la traccia inattiva, le zone di avviso e critica, l'arco attivo, la lancetta costruita come GraphicsPath, poi il valore numerico e la didascalia misurati con MeasureString e uno StringFormat.
- ✓ Racchiudi in una dichiarazione using ogni pen, brush, font e path che crei, usa le statiche del framework come Brushes.Black senza farne il dispose, e racchiudi la RotateTransform della lancetta in Graphics.BeginContainer() e EndContainer(state) così che la didascalia venga disegnata in coordinate non trasformate.
- ✓ Proteggi i casi limite: esci subito quando il rettangolo è troppo piccolo per disegnare, normalizza un intervallo invertito, limita un valore che ne esce, imposta AccessibleDescription dal setter, poi ridimensiona il browser, cambia il valore da un pulsante e annota in PaintCost.md la dimensione dell'immagine e la frequenza dei ridisegni.

Deliverable richiesti

- ✓ Controllo TelemetryGauge con proprietà Value, Minimum, Maximum e di soglia che limitano il valore e chiamano Invalidate() una sola volta per ogni cambiamento reale
- ✓ Helper puro GaugeGeometry che restituisce i limiti dell'anello, gli angoli di apertura, i punti della lancetta e il rettangolo dell'etichetta
- ✓ Handler di Paint che fa il dispose di ogni pen, brush, font e path che crea e non fa mai il dispose di e.Graphics
- ✓ Protezioni per dimensione zero, intervallo invertito e valori fuori intervallo, con un AccessibleDescription che porta la lettura
- ✓ PaintCost.md che annota la dimensione del ridisegno, la sua frequenza e quando un controllo a tema costerebbe meno

Domande di autovalutazione

- ✓ Un collega sposta i calcoli dell'anello e della lancetta direttamente dentro l'handler **Paint** perché «sono solo poche righe». Quali due cose diventano poi impossibili da fare a basso costo, e da quale delle due dipende il Modulo 6?
- ✓ Il tuo indicatore è legato a un record di macchina che porta con sé anche il nome di un operatore e una data di manutenzione. Se quei campi cambiano, il controllo deve invalidarsi? Spiega la risposta in termini di dipendenze visuali e non in termini di modello.
- ✓ Lo stesso indicatore finirà su una dashboard che ne mostra sessanta e si aggiorna ogni due secondi. Che cosa misureresti per primo, e che cosa cambieresti se le misure fossero cattive?



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.