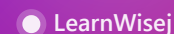


Disegno delle celle della DataGridView

Guida della lezione · Guida al lab / esame



Guida della lezione

Disegnare dentro la griglia: UserPaint a livello di cella o di colonna, DataGridViewCellPaintEventArgs con RowIndex e ColumnIndex, celle di avanzamento e sparkline che leggono le righe a basso costo, e il confronto onesto con il contenuto di cella AllowHtml su una griglia da mille righe. 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

Disegnare dentro la griglia: UserPaint a livello di cella o di colonna, DataGridViewCellPaintEventArgs con RowIndex e ColumnIndex, celle di avanzamento e sparkline che leggono le righe a basso costo, e il confronto onesto con il contenuto di cella AllowHtml su una griglia da mille righe.

1. Il problema in produzione

Il Modulo 2 ha dato a VisualOperationsStudio un `TelemetryGauge`: un controllo, una macchina, qualche centinaio di ridisegni all'ora. La griglia di operations è un animale diverso. Un capoturno apre una lista di mille macchine e vuole vedere a colpo d'occhio quali sono vicine al limite e quali sono andate alla deriva per tutta la mattina. Dei numeri in colonna non bastano, e un gauge per riga sarebbe mille controlli lato server in una sola sessione. È la griglia stessa che deve disegnare, una volta per ogni cella visibile, e smettere nel momento in cui la cella esce dallo scorrimento.

2. Concetti chiave

Concetto	Perché è importante
<code>UserPaint</code>	L'interruttore esplicito: <code>DataGridViewCell.UserPaint = true</code> per una singola cella, <code>DataGridViewColumn.UserPaint = true</code> per un'intera colonna. Senza di esso la griglia non ti chiederà mai di disegnare.
<code>CellPaint</code>	L'evento a livello di griglia sollevato per le celle disegnate a mano quando devono essere rese; ti iscrivi una volta sola sulla griglia, non una per riga, e smisti dentro l'handler.
<code>DataGridViewCellPaintEventArgs</code>	Deriva da <code>PaintEventArgs</code> , quindi porta <code>Graphics</code> e <code>ClipRectangle</code> , e aggiunge <code>RowIndex</code> e <code>ColumnIndex</code> così che un solo handler possa servire più colonne.
niente helper WinForms	Gli argomenti dell'evento documentati da Wisej.NET non espongono i metodi di comodo di WinForms <code>PaintBackground()</code> e <code>PaintContent()</code> . È il tuo handler a possedere l'intero risultato visivo di quella cella.
<code>e.ClipRectangle</code> come superficie	Il rettangolo di destinazione locale della cella. Le colonne possono essere ridimensionate e la griglia disposta in modo responsive, quindi ogni coordinata che disegni va ricavata da lì e non da una costante.
helper di rendering puri	Metodi con la forma <code>DrawProgressCell(Graphics g, Rectangle rect, double value)</code> , così l'handler si riduce a codice di instradamento e lo stesso helper potrà poi rendere dentro un bitmap off-screen.
contenuto di cella <code>AllowHtml</code>	L'alternativa nativa del browser: icone, span, label e CSS resi dal client, ed è la ragione per cui la documentazione riserva <code>CellPaint</code> agli elementi visivi troppo complessi per l'HTML.

3. Il filo della lezione

Trasformare una cella in una superficie di disegno

In una `DataGridView` di Wisej.NET nulla è disegnato a mano per default. Aderisci cella per cella o colonna per colonna impostando `UserPaint` a `true`, e solo allora la griglia solleva `CellPaint` per quella superficie. Dimenticare l'interruttore è il motivo più frequente per cui un handler apparentemente corretto non viene mai eseguito, quindi tieni vicine, nel tuo metodo di configurazione delle colonne, la riga che abilita e quella che si iscrive.

Preferisci l'ambito di colonna. Nel lab aggiungi colonne esplicite a `gridOperations` e imposti `UserPaint = true` solo sulle colonne Health e Trend, poi ti iscrivi una volta sola a `CellPaint` sulla griglia. Così hai una sola iscrizione per l'intera schermata invece di una per riga, e tutte le altre colonne — identificativo, nome macchina, sito, ultima lettura — restano normale contenuto di griglia che il browser può selezionare, cercare e scalare. Abilitare il disegno a mano per tutta la griglia è la decisione opposta: ti consegna celle che non avevi intenzione di disegnare e chiede all'handler di filtrarle a ogni ridisegno.

Che cosa ti passa l'evento, e che cosa no

Poiché `DataGridViewCellPaintEventArgs` deriva da `PaintEventArgs`, la parte di disegno è quella insegnata dal Modulo 2: `e.Graphics` è la superficie, `e.ClipRectangle` è la tua area, e fai il dispose di ogni pen, brush e path che crei lasciando in pace `e.Graphics`. La novità è lo smistamento. Sia `e.RowIndex` sia `e.ColumnIndex` hanno bisogno di una guardia prima che tu tocchi qualsiasi cosa: un `RowIndex` negativo non è una riga di dati, e una colonna per cui non hai aderito deve uscire immediatamente.

La parte compatta dell'API è quella che coglie di sorpresa. Gli argomenti dell'evento di Wisej non espongono `PaintBackground()` né `PaintContent()`, quindi un esempio WinForms che li chiama va riscritto e non compilato, e sei tu a decidere come appaiono sfondo e contenuto della cella. Prendilo come un vantaggio: quando è l'handler a produrre l'intera cella, non esiste uno stato mezzo disegnato su cui ragionare.

```
private void gridOperations_CellPaint(object sender, DataGridViewCellPaintEventArgs e)
{
    if (e.RowIndex < 0)
        return;

    var rect = Rectangle.Inflate(e.ClipRectangle, -4, -5);

    if (e.ColumnIndex == healthColumn.Index)
        DrawHealthBar(e.Graphics, rect, Convert.ToInt32(
            gridOperations.GetValue(e.ColumnIndex, e.RowIndex) ?? 0));
    else if (e.ColumnIndex == trendColumn.Index)
        DrawSparkline(e.Graphics, rect, TrendOf(e.RowIndex));
}
```

Celle di avanzamento, di stato e sparkline

Tre forme coprono la maggior parte delle griglie operative. Una cella di avanzamento riempie una traccia inattiva con un rettangolo attivo la cui larghezza è proporzionale a un valore normalizzato, e disegna un numero solo quando la colonna è abbastanza larga. Una cella di stato è una singola ellisse piena, a volte con una breve etichetta; è l'elemento visivo utile più economico della griglia e spesso è più chiaro di un'icona. Una sparkline normalizza una piccola serie sul rettangolo della cella e collega i punti con `DrawLines`, come fa la colonna Trend nel lab: il suo compito è riassumere, mai riprodurre una serie da diecimila punti dentro quaranta pixel.

Tieni i calcoli fuori dall'handler. Dai a ciascun elemento visivo un helper che riceve un `Graphics`, un `Rectangle` e i valori che gli servono, e l'handler si riduce a una guardia e due chiamate. Il renderer off-screen del Modulo 6 potrà poi chiamare lo stesso metodo per mettere la sparkline identica dentro un PNG esportato. Non dimenticare gli stati che la griglia possiede: una cella disegnata a mano deve restare leggibile anche quando la sua riga è selezionata, quindi scegli colori consapevoli della selezione oppure tieni l'elemento visivo sobrio e lascia parlare l'evidenziazione della riga. Non lasciare mai che il colore sia l'unico portatore di uno stato critico; lo stesso valore deve esistere anche come testo o come tooltip.

Quando disegnare è lo strumento sbagliato

Wisej.NET rende anche HTML dentro le celle tramite `AllowHtml`, e la documentazione colloca `CellPaint` come risposta per contenuti troppo complessi per l'HTML o per codice di disegno .NET esistente che vuoi riutilizzare. Scegli l'HTML quando l'elemento visivo si può esprimere con testo, icone, span e CSS, quando ricerca, selezione, accessibilità o scalatura del browser traggono beneficio da contenuto DOM reale, o quando le celle sono molte migliaia. Scegli il disegno quando ti serve geometria a livello di pixel, quando l'elemento visivo è compatto e cambia di rado, o quando l'HTML finirebbe per essere più complicato del codice di disegno.

Una seconda alternativa non ha nulla a che vedere con l'HTML: i tipi di cella della griglia stessa e la sua capacità di ospitare controlli. Un pulsante, una checkbox, un date picker o una combo box devono restare celle interattive vere. Il disegno è un modello di rendering, non un modello di interazione, e un pulsante disegnato ti costringe a ricostruire a mano feedback del puntatore, accesso da tastiera, focus e comandi. È per questo che il lab costruisce l'indicatore di stato due volte e ti chiede di mettere per iscritto quale manderesti in produzione.

Scorrere mille righe

Una `DataGridView` regge bene grandi insiemi di dati con il binding e le modalità virtuali, e un handler di `CellPaint` distratto annulla quel vantaggio in una sola riga. Lo scorrimento richiama l'handler per ogni nuova cella che diventa visibile, quindi qualsiasi cosa costosa viene moltiplicata per la velocità con cui l'utente trascina la scrollbar.

- Non fare mai I/O, chiamate al database o a servizi dentro l'handler.
- Non caricare immagini né creare font per cella; costruiscili una volta e riusali.
- Disegna il minor numero possibile di colonne e riduci il numero di punti delle sparkline.
- Tieni il recupero dello stato $O(1)$: indicizza la lista che hai associato, oppure leggi il valore della cella attraverso la griglia, ma non cercare mai dentro una collezione per ogni cella.
- Misura con numeri di righe realistici — cento righe non dimostrano nulla su mille.

Le mille righe generate del lab esistono proprio per quest'ultimo punto. Associale, trascina la scrollbar dall'alto in basso e confronta quanto costano le due colonne disegnate rispetto alla colonna HTML che sta loro accanto. Nelle tue note va quella misura, non una preferenza.



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:

- ✓ Abilitare il disegno a livello di cella o di colonna con UserPaint.
- ✓ Usare DataGridViewCellPaintEventArgs in sicurezza.
- ✓ Costruire celle di avanzamento, di stato e sparkline.
- ✓ Confrontare il disegno a mano con l'HTML nelle celle, i tipi di cella predefiniti, i controlli nelle celle e le immagini.
- ✓ Progettare il disegno delle celle per grandi insiemi di dati senza trasformare lo scorrimento in un collo di bottiglia del rendering.

Guida al lab / esame

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



Lab pratico

Dai a VisualOperationsStudio una griglia di operations con due colonne disegnate a mano. Aggiungi colonne esplicite a gridOperations, imposta UserPaint = true solo sulle colonne Health e Trend e gestisci CellPaint una volta sola: esci immediatamente quando e.RowIndex è negativo o e.ColumnIndex non è una delle due colonne bersaglio, poi disegna una barra di avanzamento colorata per soglia per Health e una sparkline a polilinea per Trend dentro e.ClipRectangle. Leggi i valori di ogni riga dall'oggetto MachineStatus associato invece di cercare dentro una lista nell'handler, rispetta i colori della riga selezionata e tieni l'handler privo di effetti collaterali. Associa 1.000 righe generate, scorri la griglia e poi costruisci lo stesso indicatore di stato una seconda volta come cella AllowHtml, così da poter dichiarare per iscritto quale dei due manderesti in produzione.

Punto di partenza

Apri VisualOperationsStudio come l'ha lasciato il Modulo 2: la `VisualOperationsPage` porta il controllo `TelemetryGauge` e il suo helper `GaugeGeometry`, e la pagina ha spazio per una griglia ma non ha ancora dati. Aggiungi una classe `MachineStatus` con un identificativo, un nome, un sito, una percentuale `Health` e una breve lista di letture recenti per l'andamento, più un generatore capace di produrne un migliaio.

Approccio suggerito

- ✓ Aggiungi `gridOperations` con colonne esplicite per identificativo, macchina, sito, Health e Trend, associa prima dieci righe `MachineStatus` e verifica che le colonne semplici si rendano prima che esista qualsiasi disegno.
- ✓ Imposta `UserPaint = true` solo sulle colonne Health e Trend, iscriviti una volta sola a `gridOperations.CellPaint` e apri l'handler con una guardia che esce quando `e.RowIndex` è negativo o `e.ColumnIndex` non è nessuna delle due colonne bersaglio.
- ✓ Scrivi `DrawHealthBar(Graphics, Rectangle, int)` e `DrawSparkline(Graphics, Rectangle, IReadOnlyList<double>)` come helper puri che ricavano ogni coordinata dal rettangolo che ricevono, scelgono il colore di riempimento dalle soglie di warning e critica e fanno il dispose di ogni pen e brush che creano.
- ✓ Rendi l'handler pura logica di instradamento: riduci `e.ClipRectangle` con `Rectangle.Inflate`, recupera i valori della riga in un unico passo O(1), passali all'helper e tieni leggibile la riga selezionata scegliendo colori che contrastino ancora con lo sfondo della selezione.
- ✓ Associa 1.000 righe generate, trascina la scrollbar dall'alto in basso, poi aggiungi una terza colonna di stato che usa `AllowHtml` per rendere lo stesso stato di warning o critico, scorri di nuovo e annota entrambi i risultati.

Deliverable richiesti

- ✓ `gridOperations` con colonne esplicite e `UserPaint = true` solo sulle colonne Health e Trend

- ✓ Un unico handler di CellPaint che verifica e.RowIndex e e.ColumnIndex prima di disegnare qualsiasi cosa
- ✓ Cella di avanzamento colorata per soglia e cella sparkline a polilinea disegnate dentro e.ClipRectangle
- ✓ Valori di riga letti dall'oggetto MachineStatus associato, senza ricerche, allocazioni o creazione di controlli per cella
- ✓ Una colonna di stato AllowHtml costruita in parallelo, più una nota che confronta le due soluzioni su uno scorrimento di 1.000 righe

Domande di autovalutazione

- ✓ Un collega ti passa un esempio WinForms di disegno delle celle che inizia chiamando `e.PaintBackground( ... )`. Spiega che cosa ti danno invece gli argomenti dell'evento in Wisej.NET e che cosa deve ora decidere da solo l'handler.
- ✓ Il tuo helper per la sparkline ha una larghezza di tracciato fissa a 100 pixel e sul tuo schermo sembra a posto. Descrivi due azioni dell'utente che lo rompono e che cosa dovrebbe leggere l'helper al suo posto.
- ✓ La colonna di stato HTML e la colonna Health disegnata mostrano entrambe una macchina in stato critico. Argomenta quale delle due manderesti in produzione per una griglia che il personale di supporto cerca e legge con uno screen reader, e che cosa perderesti in entrambi i casi.



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.