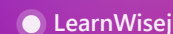


Padroneggiare la DataGridView

Guida della lezione · Guida al lab / esame



Guida della lezione

BindingSource e colonne esplicite, editor personalizzati e celle comando, modalità virtuale con CellValueNeeded e una cache lato server, e HTML nelle celle, controlli nelle celle e disegno delle celle usati con criterio. Che cosa significa per uno sviluppatore Wisej.NET — e come affrontare questo modulo.



Focus di produzione

BindingSource e colonne esplicite, editor personalizzati e celle comando, modalità virtuale con CellValueNeeded e una cache lato server, e HTML nelle celle, controlli nelle celle e disegno delle celle usati con criterio.

1. Il problema in produzione

La `DataGridView` è il controllo che più di ogni altro decide se un'applicazione gestionale sembra matura. Ordini, fatture, scadenze, log, utenti e documenti finiscono tutti in una griglia, e gli utenti giudicano l'applicazione da come quella griglia carica, scorre, si modifica e riporta lo stato. Wisej.NET offre tutto, dal popolamento manuale alla modalità virtuale al disegno delle celle; la padronanza sta nello scegliere la tecnica giusta per la quantità di dati e per il compito dell'utente.

2. Concetti chiave

Concetto	Perché è importante
BindingSource	Un intermediario lato server tra le righe del view model e la griglia, che le dà una sorgente dati stabile, la nozione di riga corrente e l'ordinamento senza accoppiare la schermata al database.
AutoGenerateColumns	Crea una colonna per ogni proprietà pubblica del tipo in binding; comodo per un prototipo, pericoloso in produzione perché una modifica al modello riorganizza la schermata senza dire niente.
colonna esplicita	Una colonna che definisci tu con binding al campo, testo dell'intestazione, larghezza, allineamento, formato, sola lettura, modalità di ordinamento e tipo di editor, così l'interfaccia resta stabile quando il modello cambia.
modalità virtuale	Con <code>VirtualMode</code> attiva, la griglia conosce solo il numero di righe e chiede ogni valore visibile tramite <code>CellValueNeeded</code> : non viene creato nulla per le righe che nessuno vede.
cache lato server	Un buffer grande quanto una pagina che alimenta <code>CellValueNeeded</code> , così lo scorrimento non colpisce il database per ogni cella; l'evento <code>DataRead</code> è il punto documentato per costruirlo e mantenerlo.
editor di cella personalizzato	Un controllo assegnato alla proprietà <code>Editor</code> di una colonna; deve riportare il proprio valore nella cella, all'occorrenza tramite <code>CellBeginEdit</code> e <code>CellEndEdit</code> se non ha un evento di cambiamento adatto.
HTML nella cella o controllo nella cella	L'HTML prodotto in <code>CellFormatting</code> con <code>AllowHtml</code> è markup di sola visualizzazione e costa quasi nulla; un controllo incorporato è un vero controllo server più un widget nel browser, con un costo per ogni cella.

3. Il filo della lezione

Tre modi di popolare, uno adatto alla produzione

La griglia può mostrare dati con o senza una sorgente dati sottostante. L'accesso diretto a `Rows` e `Columns` è il più semplice: aggiungi colonne e righe e possiedi ogni valore; va bene per una tabellina statica come una tabella di stati. Il binding con `DataSource` e `DataMember` popola la griglia da una lista, da un `BindingSource` o da un data set, ed è la modalità per i dati di business. Il terzo modo è la generazione dinamica delle colonne: con `AutoGenerateColumns` a true la griglia ricostruisce le colonne ogni volta che cambia la sorgente dati.

Le colonne generate vanno bene per un prototipo di dieci minuti e diventano una trappola per una schermata che va in produzione: aggiungi una proprietà a `OrderRow`, rinominane un'altra, e la griglia segue il modello invece del progetto. La griglia Orders della Operations Console definisce quindi ogni colonna esplicitamente: `DataPropertyName`, `HeaderText`, `Width`, allineamento e formato in `DefaultCellStyle`, `ReadOnly` dove il valore è informativo, `SortMode` e il tipo di colonna che porta con sé l'editor giusto. Il binding in sé resta piccolo:

```
private readonly BindingSource ordersSource = new BindingSource();

private void BindOrders(IEnumerable<OrderRow> rows)
{
    this.ordersGrid.AutoGenerateColumns = false;    // le colonne si progettano, non si
    scoprono
    this.ordersSource.DataSource = rows.ToList();
    this.ordersGrid.DataSource = this.ordersSource;
}

private void ordersGrid_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)
{
    if (ordersGrid.Columns[e.ColumnIndex].Name != "StatusColumn") return;
    var status = System.Net.WebUtility.HtmlEncode(Convert.ToString(e.Value));
    e.Value = $"<span class='status-badge'>{status}</span>";
    e.FormattingApplied = true;
}
```

Disattiva `AutoGenerateColumns` appena la schermata è stabile, e metti in binding una collezione di view model invece di un'entità: un view model è modellato per la visualizzazione e porta con sé il testo dello stato, un totale già formattato e un ID senza far trapelare dettagli di persistenza.

Colonne predefinite, editor personalizzati e celle comando

I valori tipizzati meritano colonne tipizzate. I tipi di colonna testo, check box, combo box, pulsante, link e immagine coprono quasi tutta la griglia Orders, e Wisej.NET aggiunge colonne costruite sugli editor di data, numerici e con maschera del Modulo 2. Una cella comando è una colonna pulsante o link: la colonna *Ship* del lab solleva `CellContentClick`, il gestore legge l'ID della riga dall'elemento in binding e chiama il servizio. Nella cella non vive nulla dell'ordine.

Quando nessun editor predefinito è adatto, assegna un controllo alla proprietà `Editor` della colonna; l'esempio classico è un `MonthCalendar` su una cella di data di scadenza. L'avvertenza importante: l'editor deve aggiornare correttamente il valore della cella. La griglia si aspetta `TextChanged` o un evento di cambiamento equivalente; se il tuo controllo non lo solleva, gestisci `CellBeginEdit` per copiare il valore della cella nell'editor e `CellEndEdit` per riportare indietro il valore modificato. Prova il giro completo con Tab, Invio e un clic altrove: un editor che non conferma mai è il bug classico degli editor personalizzati.

Modalità virtuale, CellValueNeeded e la cache

I grandi volumi richiedono un approccio diverso. Caricare 100.000 ordini perché il database riesce a restituirli non è una strategia di interfaccia: nessuno può esaminare tutte quelle righe, e ciascuna costa memoria sul server e rendering nel browser. Filtra prima, offri ricerca e paginazione lato server, e riserva la modalità virtuale al percorso in cui l'utente deve davvero scorrere un risultato molto grande. Lì imposti `VirtualMode` a true, dai alla griglia il suo `RowCount` e le colonne, e gestisci `CellValueNeeded` per fornire ogni valore su richiesta.

L'evento da solo non basta. Un gestore che interroga il database dentro `CellValueNeeded` viene eseguito una volta per ogni cella visibile, quindi un solo scorrimento può produrre centinaia di query. La `OrderCache` del lab tiene la pagina corrente sul server: `CellValueNeeded` le chiede riga e colonna, lei

risponde dalla memoria quando c'è e recupera un'intera pagina quando manca. La documentazione di Wisej.NET indica l'evento `DataRead` come il punto in cui costruire e mantenere quella cache, perché ti dice quale blocco di righe la griglia sta per leggere. Fai il profiling con volumi realistici, non con le venti righe del servizio dimostrativo, e chiediti che cosa si rompe per primo a dieci volte i dati.

HTML nelle celle, controlli nelle celle, disegno delle celle e griglia composta

L'HTML nelle celle è la migliore via di mezzo per la ricchezza visiva. Se uno stato ha bisogno di un badge, di un'icona, di una barra di avanzamento o di più frammenti di testo, imposta `AllowHtml` e produci il markup in `CellFormatting`, come nel frammento qui sopra; lo stato grezzo resta nel modello e l'HTML è solo presentazione. Trattalo come tale: codifica il testo dell'utente prima di inserirlo nel markup, così il nome di un cliente non diventa mai HTML eseguibile, e tieni la validazione di business fuori dal gestore di formattazione.

I controlli nelle celle sono più potenti e più pericolosi. Wisej.NET può mettere un controllo vero dentro una cella, anche esteso su più celle: un pulsante in una riga, una track bar su più colonne, una mini anteprima. Ognuno è un controllo server più un widget nel browser, quindi un controllo in ogni cella di una griglia da 10.000 righe sono 10.000 controlli. Usali per la riga selezionata, per righe di riepilogo, per un'area comandi o per una griglia piccola. Il disegno delle celle sta all'estremo opposto della scala: attiva il disegno personalizzato sulla cella e gestisci `CellPaint` quando la visualizzazione richiede un disegno su misura o riusa codice di disegno .NET esistente, non per stili che temi, stili di cella o HTML già coprono.

Infine, la `DataGridView` componibile si comporta come un contenitore e può ospitare controlli collegati come un pannello di filtri, una striscia di stato, un navigatore o un albero, in dock dentro o intorno a sé e ordinati con `BringToFront` e `SendToBack`. Così l'API completa della griglia resta visibile invece di essere nascosta dietro uno UserControl; la striscia di filtri e quella di stato del lab si costruiscono in questo modo.

- Colonne esplicite e un BindingSource o un view model per ogni schermata di produzione; `AutoGenerateColumns` disattivato appena la schermata è stabile.
- Filtra prima di visualizzare; modalità virtuale più cache per gli insiemi davvero grandi.
- HTML nelle celle prima dei controlli incorporati; controlli incorporati con parsimonia; disegno delle celle solo quando stili, HTML e controlli normali non bastano.
- Fai il profiling con dati realistici, non con dati di esempio.



Errori tipici

Lasciare `AutoGenerateColumns` attivo in produzione. Caricare tutti i record all'avvio. Interrogare il database da `CellValueNeeded` per ogni cella senza una cache. Incorporare controlli in ogni riga di una griglia enorme. Mescolare validazione di business e formattazione visiva. Trattare una griglia come una tabella di database invece che come una vista orientata al compito.



La regola da libreria di controlli

Scegli il controllo nativo più semplice che esprime il compito dell'utente prima di scrivere codice personalizzato. I controlli nativi ti danno già supporto a design time, temi, eventi e stato sul server; il passo successivo è un extender o uno UserControl, e solo quando nessun controllo nativo è adatto si costruisce un widget JavaScript.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Collegare la DataGridView a un BindingSource e controllare AutoGenerateColumns in modo consapevole.
- ✓ Usare colonne predefinite ed editor personalizzati per valori tipizzati e celle comando.
- ✓ Realizzare la modalità virtuale, CellValueNeeded, i concetti di DataRead e caching e i limiti di performance sui grandi volumi.
- ✓ Usare con criterio HTML nelle celle, controlli nelle celle, disegno delle celle e griglie composte.

Guida al lab / esame

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



Lab pratico

Costruisci la griglia Orders di Operations Console: una DataGridView collegata a un BindingSource con colonne esplicite e AutoGenerateColumns sotto controllo, una cella di stato resa con AllowHtml e CellFormatting, una colonna con editor personalizzato o comando, una versione in modalità virtuale o con cache per il percorso a grandi volumi, e una striscia di filtri e di stato composta dentro o intorno alla griglia.

Punto di partenza

Apri la soluzione `OperationsConsole` del Modulo 4 e seleziona la pagina segnaposto DataGridView nello shell. L'esploratore documenti, il `CustomerEditor` e lo shell di navigazione esistono già; la pagina Orders è ancora una `TabPage` vuota, e un piccolo `OrderService` in memoria che restituisce view model `OrderRow` basta ad alimentare ogni passo di questo lab.

Approccio suggerito

- ✓ Aggiungi `ordersGrid` e un BindingSource `ordersSource`, imposta `AutoGenerateColumns` a false e definisci esplicitamente ogni colonna: numero d'ordine, cliente, data di scadenza, totale, stato — con testo dell'intestazione, larghezza, allineamento, formato e sola lettura scelti apposta.
- ✓ Marca la colonna dello stato con `AllowHtml` e gestisci `CellFormatting` per trasformare lo stato grezzo in un badge con il testo codificato; lascia intatto il valore del modello e imposta `FormattingApplied`.
- ✓ Aggiungi un editor MonthCalendar sulla colonna della data di scadenza tramite la proprietà `Editor` della colonna (con `CellBeginEdit` / `CellEndEdit` se il valore non fa il giro completo) oppure una colonna pulsante il cui `CellContentClick` chiama il servizio passando l'ID dell'ordine.
- ✓ Costruisci il percorso a grandi volumi: `VirtualMode` a true, `RowCount` dal servizio, una `OrderCache` che serve `CellValueNeeded` da una pagina in memoria e ne recupera una nuova quando manca, usando `DataRead` per precaricare il blocco che la griglia sta per mostrare.
- ✓ Componi la griglia: metti in dock una striscia di filtri (casella di ricerca, ComboBox dello stato, pulsante Applica) sopra o dentro la griglia e una striscia di stato sotto che mostra il numero di righe, l'ordine selezionato e l'ultimo aggiornamento; ordinali con `BringToFront` e `SendToBack`.

Deliverable richiesti

- ✓ DataGridView Orders con colonne esplicite collegate a un BindingSource
- ✓ Cella di stato resa con AllowHtml e CellFormatting
- ✓ Una colonna con editor personalizzato o comando
- ✓ Versione della griglia in modalità virtuale o con cache per grandi volumi
- ✓ Striscia di filtri e di stato composta dentro o intorno alla griglia

Domande di autovalutazione

- ✓ Se il prossimo sprint `OrderRow` guadagna due nuove proprietà, che cosa cambia nella schermata con le colonne generate, e che cosa cambia con le tue colonne esplicite?
- ✓ Il tuo gestore di `CellValueNeeded` funziona con 200 ordini. Che cosa succede al database quando un utente trascina la barra di scorrimento attraverso 200.000 righe, e quale parte del tuo progetto assorbe il colpo?
- ✓ Perché il badge dello stato va in `CellFormatting` e non nel modello `OrderRow`, e che cosa andrebbe storto se il nome di un cliente finisse in quell'HTML senza essere codificato?



La regola da libreria di controlli

Scegli il controllo nativo più semplice che esprime il compito dell'utente prima di scrivere codice personalizzato. I controlli nativi ti danno già supporto a design time, temi, eventi e stato sul server; il passo successivo è un extender o uno UserControl, e solo quando nessun controllo nativo è adatto si costruisce un widget JavaScript.