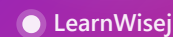


Caricare i dati in BindingSource, DataGridView e controlli di lookup

Guida della lezione · Guida al lab / esame



Guida della lezione

Esecuzione asincrona delle query LINQ, proiezione senza tracking in elementi di lista, BindingSource come collegamento verso l'interfaccia, mai un IQueryable collegato direttamente, flag di caricamento attorno ai gestori async e ComboBox di lookup con DisplayMember e ValueMember. Che cosa significa per uno sviluppatore Wisej.NET che lavora con database reali — e come affrontare questo modulo.



Focus di produzione

Esecuzione asincrona delle query LINQ, proiezione senza tracking in elementi di lista, BindingSource come collegamento verso l'interfaccia, mai un IQueryable collegato direttamente, flag di caricamento attorno ai gestori async e ComboBox di lookup con DisplayMember e ValueMember.

1. Il problema in produzione

Gli operatori di Support Desk hanno bisogno di una griglia dei ticket che cerchi rapidamente senza trascinare l'intero database nella memoria del server. In un'applicazione Wisej.NET ogni sessione tiene sul server i propri form, le proprie BindingSource e le proprie righe selezionate, quindi una schermata che carica migliaia di entità **Ticket** tracciate per utente moltiplica quel costo per il numero di browser aperti. Questo modulo costruisce il browser dei ticket nel modo giusto: componi la query, eseguila in modo asincrono, materializza una piccola pagina proiettata e passa quella lista all'interfaccia tramite una BindingSource.

2. Concetti chiave

Concetto	Perché è importante
metodo di esecuzione asincrono	<code>ToListAsync</code> , <code>CountAsync</code> e <code>SingleOrDefaultAsync</code> sono le chiamate che inviano davvero l'SQL; attenderle con <code>await</code> lascia libero il thread del server mentre il database lavora.
proiezione	Una <code>Select</code> in un record <code>TicketListItem</code> restituisce esattamente le colonne che la griglia mostra, invece dell'entità Ticket con il suo grafo di Customer, Agent e Comments.
query senza tracking	<code>AsNoTracking</code> dice al Context di non fare lo snapshot dei risultati: meno memoria, nessun aggiornamento accidentale e una lista indipendente da qualsiasi unit of work di modifica.
BindingSource	Il componente Wisej.NET tra una collezione di dati e i controlli collegati; gestisce l'elemento corrente, la navigazione e le operazioni sulla lista e mantiene sincronizzati la griglia e i controlli di dettaglio.
IQueryable	Una ricetta per una query sul database, non una lista stabile: deve essere eseguita e materializzata prima di assegnare qualsiasi cosa all'interfaccia.
protezione di caricamento	Un flag, pulsanti disabilitati e una label di stato attorno a ogni chiamata al database attesa con <code>await</code> , ripristinati nel <code>finally</code> , così gira un'operazione alla volta e la pagina resta utilizzabile dopo un errore.
binding dei lookup	Una ComboBox con <code>DisplayMember</code> per il nome mostrato e <code>ValueMember</code> per la chiave memorizzata, caricata prima che parta la prima ricerca o che venga assegnato il modello di modifica.

3. Il filo della lezione

Componi la query, poi eseguila una volta sola

Un `IQueryable<Ticket>` descrive un lavoro che il database non ha ancora fatto: ogni `Where` , `OrderBy` , `Skip` , `Take` e `Select` confluisce in quella descrizione, e niente viene eseguito finché non si attende un metodo di esecuzione. Il `SearchTicketsAsync` di Support Desk parte da `db.Tickets.AsNoTracking()` , aggiunge un `Where` per ogni filtro che l'operatore ha effettivamente impostato, ordina per `UpdatedAt` , pagina con `Skip` e `Take` , proietta, e solo allora attende `ToListAsync` . Un `CountAsync` separato sulla query filtrata, prima di tagliare la pagina, fornisce il totale: due istruzioni SQL modellate dagli stessi filtri.

Le varianti asincrone contano perché un gestore di eventi Wisej.NET gira su un thread del server condiviso con tutte le sessioni: attendere `ToListAsync` rilascia quel thread mentre l'SQL è in esecuzione, e `SingleOrDefaultAsync` fa lo stesso per un singolo ticket caricato per chiave. Il metodo di servizio, abbreviato:

```
await using var db = await _dbFactory.CreateDbContextAsync(token);
var query = db.Tickets.AsNoTracking();
if (!string.IsNullOrEmpty(criteria.Text))
    query = query.Where(t => t.Number.Contains(criteria.Text) ||
t.Title.Contains(criteria.Text));
if (criteria.CustomerId is not null)
    query = query.Where(t => t.CustomerId == criteria.CustomerId.Value);

var total = await query.CountAsync(token); // totale prima di tagliare la pagina
var page = await query
    .OrderByDescending(t => t.UpdatedAt)
    .Skip(criteria.PageIndex * criteria.PageSize).Take(criteria.PageSize)
    .Select(t => new TicketListItem(t.Id, t.Number, t.Title, t.Customer.Name, /* ... */
t.UpdatedAt))
    .ToListAsync(token); // l'unica riga che esegue la query
della pagina
return new PagedResult<TicketListItem>(page, total);
```



Errore tipico: chiamare ToList prima di filtrare o paginare

Un `ToListAsync` seguito da `Where`, `Skip` o `Take` trascina in rete l'intera tabella dei ticket e poi la filtra e la pagina nella memoria del server. Componi ogni clausola sull' `IQueryable`; fai di `ToListAsync` l'ultima chiamata.

BindingSource: il collegamento tra una lista e la griglia

Wisej.NET supporta due tipi di binding: il binding semplice collega una proprietà di un controllo a una proprietà dell'elemento corrente (`TextBox.Text` a `Title`), e il binding complesso collega un controllo di lista, una `DataGridView` o una `ComboBox`, a un'intera collezione. `BindingSource` è il collegamento per entrambi: avvolge la collezione, tiene traccia dell'elemento corrente, supporta la navigazione tramite `Position` e `MoveNext`, espone le operazioni sulla lista e solleva le notifiche di cambiamento che tengono al passo ogni controllo collegato. La griglia si collega alla `BindingSource`; il `DataSource` della `BindingSource` è la lista materializzata.

Quella lista può essere una semplice `List<TicketListItem>` o una collezione di DTO, una `BindingList<T>` quando i controlli devono reagire ad aggiunte o rimozioni di elementi, oppure una piccola collezione di lookup per una `ComboBox`. Nel browser dei ticket, `ticketsDataGridView.DataSource = ticketBindingSource` viene impostato una sola volta nel costruttore con `AutoGenerateColumns = false`, e il `DataPropertyName` di ogni colonna indica una proprietà di `TicketListItem`. Dopo ogni ricerca cambia solo il `DataSource` della `BindingSource`; la griglia e le sue colonne restano come sono state progettate.



Errore tipico: collegare un IQueryable direttamente a una griglia

Assegnare `db.Tickets` o la query stessa a un controllo consegna all'interfaccia una ricetta viva legata a un Context che sta per essere eliminato. Esegui con `ToListAsync`, dai la lista alla `BindingSource` e collega la griglia alla `BindingSource`, mai a un `DbSet`.

Proiezione senza tracking per le griglie, entità tracciate per gli editor

La griglia ha bisogno di sette colonne, non di Description, RowVersion, della collezione Comments o di Customer e Agent come oggetti completi. Caricarli comunque, cioè inondare l'interfaccia di entità, costa byte in rete, memoria di sessione e uno snapshot del change tracker per ogni entità, e apre la porta a un bug sottile: una griglia piena di entità tracciate sembra modificabile, e un `SaveChangesAsync` sfuggito scriverebbe tutto ciò che è stato toccato. Proiettare con `Select` in `TicketListItem` e marcare la query `AsNoTracking` elimina tutto questo; le proprietà di navigazione restano utilizzabili dentro la proiezione (`t.Customer.Name`, `t.Agent == null ? null : t.Agent.DisplayName`) ed EF Core le traduce in join.

Il flusso di modifica è l'immagine speculare: l'editor del Modulo 4 crea il proprio Context dalla factory, carica l'entità tracciata per chiave con `SingleOrDefaultAsync`, la collega o la mappa, e salva tramite quello stesso Context. L'elemento di lista nella griglia non viene mai collegato al Context, aggiornato o salvato; quando l'editor si chiude, la pagina riesegue la ricerca. Così la lista dell'interfaccia resta indipendente dalla unit of work di modifica.



Errore tipico: caricare interi grafi di entità per le schermate di lista

Una catena di chiamate `Include` seguita da `ToList` materializza ticket, clienti, agenti e commenti che la griglia non mostra mai, una volta per sessione. Proietta nell'elemento di lista; riserva i grafi tracciati e inclusi all'editor che li salverà.

Ricerca, paginazione, lookup e feedback di avanzamento

I gestori di eventi Wisej.NET possono essere `async void`, quindi `searchButton_Click` azzerà `_pageIndex` e attende `LoadTicketsAsync`; i pulsanti di paginazione fanno lo stesso con l'indice aggiornato. È quel metodo a contenere la protezione di caricamento. Se `_loading` è già true ritorna subito; altrimenti imposta il flag, disabilita i tre pulsanti, scrive "Loading tickets..." nella label di stato, costruisce un `TicketSearchCriteria` dai controlli, attende una sola chiamata al servizio, assegna `result.Items.ToList()` alla BindingSource e riporta "Showing 50 of 312 tickets". Il blocco `catch` mostra un `AlertBox` comprensibile; il `finally` riabilita i pulsanti e azzerà il flag su ogni percorso. EF Core non supporta operazioni parallele su un Context; la protezione lo garantisce dal lato dell'interfaccia.

Un lavoro davvero lungo, un'esportazione o una riassegnazione massiva, non dovrebbe tenere occupata la richiesta: `Application.StartTask` lo esegue in background e `Application.Update` spinge le modifiche dell'interfaccia verso il browser fuori banda, con il DbContext creato dentro l'attività ed eliminato lì, mai preso in prestito dalla pagina.

Stato, categoria, agente e cliente sono piccole liste di lookup. Caricale nel gestore `Load` della pagina, prima della prima ricerca: collega ogni ComboBox alla sua lista, imposta `DisplayMember` sul nome e `ValueMember` sulla chiave, e leggi `SelectedValue` quando costruisci i criteri così la query filtra per `CustomerId`, non per testo visualizzato. In un editor, collega `SelectedValue` alla chiave esterna del modello di modifica e popola il lookup prima di assegnare il modello di modifica alla sua BindingSource; un SelectedValue senza un elemento corrispondente si risolve silenziosamente in nulla. Quando un lookup supera qualche centinaio di righe, dagli una propria ricerca o paginazione.



La regola del data binding

Tieni separati i due cicli di vita. Form Wisej.NET, BindingSource e riga selezionata vivono per la sessione; un DbContext vive per un'operazione. Crealo dalla factory, interroga o salva, eliminalo, e non lasciarlo mai diventare stato di sessione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Usare metodi di esecuzione LINQ asincroni come ToListAsync e SingleOrDefaultAsync.
- ✓ Collegare liste, BindingList<T>, DTO e collezioni di lookup tramite BindingSource.
- ✓ Usare la proiezione senza tracking per le griglie di sola lettura e le entità tracciate per i flussi di modifica.
- ✓ Implementare ricerca, ordinamento, paginazione, binding delle ComboBox di lookup e feedback di avanzamento.

Guida al lab / esame

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



Lab pratico

Costruisci il browser dei ticket di Support Desk: i record `TicketListItem` e `TicketSearchCriteria`, un metodo `SearchTicketsAsync` che compone `AsNoTracking`, `Where`, `OrderBy`, `Skip`, `Take` e `Select` prima di `ToListAsync`, una `DataGridView` collegata tramite una `BindingSource` a `TicketListItem`, `ComboBox` di lookup per stato e cliente caricate prima della prima ricerca, i pulsanti `Search`, `Next Page` e `Previous Page`, la visualizzazione del conteggio totale e della dimensione di pagina, e una protezione di caricamento che mantiene l'interfaccia utilizzabile dopo un errore.

Punto di partenza

Apri la soluzione Support Desk Data Console come l'hai lasciata dopo il Modulo 2: i progetti Web, Data, Services e Tests, `SupportDeskContext` registrato tramite `AddDbContextFactory`, la migrazione `InitialCreate` applicata e i dati iniziali al loro posto (cinque clienti, tre agenti, le categorie e almeno cinquanta ticket). La pagina Tickets è ancora vuota a parte una label di titolo, e `TicketQueryService` contiene solo la query di conteggio del Modulo 1.

Approccio suggerito

- ✓ In `SupportDesk.Services` aggiungi i record `TicketListItem`, `TicketSearchCriteria` e `PagedResult<T>`, poi scrivi `SearchTicketsAsync`: un Context dalla factory, `AsNoTracking`, un `Where` per ogni filtro impostato, `CountAsync` per il totale, poi `OrderByDescending(t => t.UpdatedAt)`, `Skip`, `Take`, `Select` e `ToListAsync`.
- ✓ Su `TicketBrowserPage` aggiungi una `BindingSource` e una `DataGridView`, imposta `AutoGenerateColumns` a false, definisci le colonne con `DataPropertyName` per Number, Title, CustomerName, AgentName, Status, DueDate e UpdatedAt, e collega la griglia alla `BindingSource` nel costruttore.
- ✓ Nel gestore `Load` della pagina popola le `ComboBox` di stato e cliente con `DisplayMember` e `ValueMember` impostati, compresa una voce "All" che produce un criterio null, prima che parta la prima ricerca.
- ✓ Scrivi `LoadTicketsAsync` con la protezione di caricamento: uscita anticipata su `_loading`, pulsanti disabilitati, una label di stato, una sola chiamata al servizio attesa con `await`, la lista materializzata assegnata alla `BindingSource`, un `AlertBox` nel `catch`, e pulsanti e flag ripristinati nel `finally`; collega a questo metodo `Search` (con azzeramento dell'indice di pagina), `Next Page` e `Previous Page`.
- ✓ Mostra nella label di stato il numero di righe restituite, il conteggio totale e la dimensione di pagina, disabilita `Previous Page` sulla prima pagina e `Next Page` sull'ultima, poi punta la stringa di connessione a un server fermo per dimostrare che la pagina resta utilizzabile dopo un errore.

Deliverable richiesti

- ✓ Record `TicketListItem` e `TicketSearchCriteria` con `SearchTicketsAsync`

- ✓ DataGridView collegata a TicketListItem tramite una BindingSource
- ✓ Filtri e paginazione composti nella query così che il lavoro lo faccia il database
- ✓ ComboBox di lookup caricate prima della prima ricerca, che mostrano i nomi e memorizzano le chiavi
- ✓ Pulsanti Search, Next Page e Previous Page con un conteggio totale e una protezione di caricamento

Domande di revisione per lo studente

- ✓ Quale ciclo di vita hai scelto per il DbContext dentro `SearchTicketsAsync`, e che cosa si romperebbe se la pagina tenesse vivo un solo Context per tutta la sua durata?
- ✓ Dopo una ricerca, quali dati vengono tenuti sul server per questa sessione e quali restano solo nel database? Quanto costa un secondo operatore in contemporanea?
- ✓ Se il lookup dei clienti crescesse fino a cinquantamila righe, quali parti della pagina cambieresti e quali parti del servizio di query resterebbero esattamente come sono?



La regola del data binding

Tieni separati i due cicli di vita. Form Wisej.NET, BindingSource e riga selezionata vivono per la sessione; un DbContext vive per un'operazione. Crealo dalla factory, interroga o salva, eliminalo, e non lasciarlo mai diventare stato di sessione.