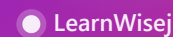


Architettura: EF Core dentro un'applicazione Wisej.NET

Guida della lezione · Guida al lab / esame



Guida della lezione

Dove si colloca EF Core in un'app Wisej.NET con stato: i cicli di vita di sessione, interfaccia, richiesta e unit of work, una DbContextFactory con un Context per operazione, il collegamento verso la DI di Microsoft e le regole di sicurezza tra sessioni. Che cosa significa per uno sviluppatore Wisej.NET che lavora con database reali — e come affrontare questo modulo.



Focus di produzione

Dove si colloca EF Core in un'app Wisej.NET con stato: i cicli di vita di sessione, interfaccia, richiesta e unit of work, una DbContextFactory con un Context per operazione, il collegamento verso la DI di Microsoft e le regole di sicurezza tra sessioni.

1. Il problema in produzione

Un'applicazione Wisej.NET sembra WinForms perché form, pagine, controlli, componenti `BindingSource` e gestori di eventi vivono tutti sul server, e questa familiarità invita a una scorciatoia pericolosa: trattare un `DbContext` come un altro oggetto server di lunga durata. Anche EF Core gira sul server, ma non è un contenitore di stato dell'interfaccia; è una breve unit of work il cui change tracker non è thread-safe né pensato per essere condiviso. Molte sessioni del browser apriranno la Support Desk Data Console contemporaneamente, quindi la prima competenza di questo corso è tenere separati i cicli di vita dell'interfaccia e quello del database, prima ancora di collegare qualsiasi griglia.

2. Concetti chiave

Concetto	Perché è importante
stato di sessione	Lo stato lato server per ogni sessione del browser che Wisej.NET mantiene vivo: pagine aperte, riga selezionata, filtri, messaggi di validazione e modifiche in sospeso, raggiungibile tramite <code>Application.Session</code> e l'albero dei controlli attivo.
ciclo di vita degli oggetti dell'interfaccia	Quanto esiste una <code>Page</code> , un <code>Form</code> o una <code>BindingSource</code> , dalla costruzione finché l'utente non la chiude o la sessione termina; spesso ore, molto più di qualsiasi chiamata al database.
ciclo di vita di richiesta/thread	Un evento del browser o una continuazione di <code>await</code> sul server; il thread che ha avviato un gestore di click non è necessariamente quello che lo termina.
unit of work	Il ciclo di vita di un <code>DbContext</code> : crealo, interroga o modifica le entità, chiama <code>SaveChangesAsync</code> , eliminalo; secondi, non sessioni.
DbContextFactory	Il servizio <code>IDbContextFactory<T></code> registrato da <code>AddDbContextFactory</code> che fornisce un Context nuovo e configurato correttamente ogni volta che un'operazione ne ha bisogno.
ponte verso la DI di Microsoft	La registrazione dell' <code>IServiceProvider</code> di ASP.NET Core in <code>Application.Services</code> , così le pagine Wisej.NET possono risolvere gli stessi servizi configurati allo startup.
fuga di dati tra sessioni	Un oggetto mutabile statico o condiviso (un Context, un change tracker, una lista collegata, un'entità selezionata) che la sessione di un utente può leggere dopo che la sessione di un altro utente l'ha scritto.

3. Il filo della lezione

Quattro cicli di vita su un solo server

Tutto ciò che sta in un processo Wisej.NET appartiene a uno di quattro cicli di vita, e la maggior parte dei bug di data binding nasce dal mettere un oggetto in quello sbagliato. Lo stato di sessione vive quanto la sessione del browser: l'identità dell'utente, il browser dei ticket aperto, i filtri di ricerca correnti. Il ciclo di vita degli oggetti dell'interfaccia appartiene a una specifica `Page` o a un `Form` e ai componenti che possiede, come la `BindingSource` dietro la griglia dei ticket e la riga selezionata; un editor modale esiste solo finché il dialogo è aperto. Il ciclo di vita della richiesta è un singolo round trip: il browser segnala un click, il server esegue il gestore, e qualsiasi `await` al suo interno può riprendere su un thread diverso del thread pool, quindi un gestore non può presupporre l'affinità di thread. Il ciclo di vita della unit of work è quello di EF Core: un `DbContext` viene creato, interroga o modifica una manciata di entità, salva e viene eliminato.

Si annidano male. Una sessione sopravvive a migliaia di richieste, un form sopravvive a molte chiamate al database e un Context non dovrebbe sopravvivere a nessuno dei due. Un Context tenuto in una sessione o in un form traccia ogni entità che abbia mai caricato, cresce finché la sessione non termina e diventa una seconda copia obsoleta del database che l'utente crede aggiornata. Due operazioni asincrone sullo stesso Context fanno lanciare un'eccezione a EF Core, perché un Context non è thread-safe. Sono queste le

modalità di errore dietro la prima domanda del modulo: perché un `DbContext` statico o condiviso è pericoloso.

Registrare EF Core e fare da ponte tra i due container

Wisej.NET ha un proprio sistema di servizi e può anche lavorare con l' `IServiceProvider` di Microsoft. Per EF Core la DI di Microsoft è la sede naturale: `AddDbContextFactory<SupportDeskContext>` (oppure `AddPooledDbContextFactory` quando conta il ricambio delle connessioni) registra una factory che conosce il provider, la stringa di connessione dalla configurazione e la diagnostica riservata allo sviluppo. I servizi che usano la factory, come il `TicketQueryService` che il lab risolve da una pagina, vanno nello stesso container. Dopo `builder.Build()`, una sola riga consegna il provider di Microsoft a Wisej.NET così i contenitori di interfaccia di primo livello possono risolvere quei servizi, con `Application.Services.GetService<T>()` oppure tramite property injection con l'attributo `[Inject]` su una `Page` o un `Form`.

```
var builder = WebApplication.CreateBuilder(args);
var cs = builder.Configuration.GetConnectionString("SupportDesk")
    ?? throw new InvalidOperationException("Missing connection string 'SupportDesk'.");

builder.Services.AddDbContextFactory<SupportDeskContext>(options =>
{
    options.UseSqlServer(cs, sql => sql.EnableRetryOnFailure());
    if (builder.Environment.IsDevelopment())
        options.EnableDetailedErrors(); // logging dei dati sensibili: solo sviluppo
    locale
});
builder.Services.AddTransient<TicketQueryService>();

var app = builder.Build();
Application.Services.AddService<IServiceProvider>(app.Services); // il collegamento verso Wisej.NET
```

Una volta installato il collegamento vale una sola regola: registra ogni servizio applicativo tramite la DI di Microsoft e non mescolare i container, altrimenti ti ritrovi con due oggetti dove te ne aspettavi uno. La stringa di connessione arriva dalla configurazione, mai da una costante su una schermata, e `EnableSensitiveDataLogging` è un interruttore per lo sviluppo locale che non deve mai arrivare nei log di produzione. La factory di design time nel progetto dati esiste solo perché gli strumenti `dotnet ef` possano costruire un Context senza avviare l'applicazione web.

Un Context per operazione, per form di modifica o dietro un servizio

Il default più sicuro per le schermate dati Wisej.NET è la factory più un Context per operazione. Una ricerca nella griglia, il caricamento di un lookup, un comando di salvataggio e uno di eliminazione creano ed eliminano ciascuno il proprio Context. È in linea con le indicazioni di EF Core per i framework di interfaccia lato server con stato, dove i componenti vivono a lungo e un Context con scope di richiesta non coincide con il ciclo di vita dell'interfaccia. Nella Support Desk Data Console la forma è un metodo di servizio:

```
await using var db = await _factory.CreateDbContextAsync(); seguito da await
```

`db.Tickets.CountAsync()`, che è esattamente la prima query asincrona che il lab esegue protetta da un flag di caricamento e da un messaggio di errore, senza ancora collegare nulla.

Un Context più longevo è accettabile in un solo punto: dentro un editor modale, quando l'utente apre un ticket, modifica entità tracciate, naviga qualche piccola collezione correlata come i suoi commenti, e salva o annulla come un'unica unità. Anche in quel caso è privato di quell'istanza dell'editor, protetto in modo che due operazioni non vi girino in parallelo, ed eliminato nel percorso di dispose del form, mai in un servizio condiviso, in un campo statico o in una cache a livello di applicazione. La terza opzione è il confine di repository o di servizio: le pagine parlano con `TicketQueryService` e `TicketCommandService`, che possiedono la factory, così l'interfaccia non vede mai un Context. Quel confine cresce nel corso dei lab successivi, ed è il motivo per cui il lab risolve dalla pagina un servizio di query, non un Context.

Sicurezza tra sessioni: che cosa si può condividere e che cosa mai

Ogni sessione del browser ha i propri oggetti dell'interfaccia, la propria riga selezionata, i propri messaggi di validazione, filtri e modifiche in sospeso; sono questioni di sessione o di form. Le entità EF Core, le istanze di `DbContext` e i change tracker non devono attraversare le sessioni. Una cache statica può contenere dati di riferimento immutabili, come le categorie dei ticket una volta caricate, oppure delegati di query compilate, perché nessuno li modifica. Non deve mai contenere un `DbContext` mutabile, una `BindingSource` o un'entità selezionata, perché un campo statico è condiviso da tutte le sessioni sul server e l'ultimo che scrive vince in silenzio.

La stessa regola vale per l'elaborazione in background: un'attività avviata con `Application.StartTask` crea il proprio Context dentro l'operazione e aggiorna l'interfaccia tramite i meccanismi di Wisej.NET consapevoli della sessione, invece di prendere in prestito un Context creato da una pagina. Dentro un gestore, ogni chiamata a EF Core viene attesa prima di riusare lo stesso Context, i pulsanti sono disabilitati mentre un caricamento o un salvataggio è in corso, e un messaggio di stato comprensibile sostituisce l'eccezione grezza; il flag di caricamento del lab è la versione più piccola di questa disciplina.



Errori tipici

Registrare il `DbContext` come servizio Wisej.NET globale condiviso, così ogni sessione traccia entità nello stesso oggetto. Condividere entità o una `BindingSource` tra sessioni tramite un campo statico o una cache. Avviare operazioni EF Core parallele sullo stesso Context, come due query lanciate senza attendere la prima. Tenere vivo un Context in una pagina perché sembra più veloce, finché il suo change tracker non contiene ogni ticket mai aperto. Mescolare servizi tra il registro di Wisej.NET e la DI di Microsoft dopo aver installato il collegamento.



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:

- ✓ Spiegare la differenza tra stato di sessione Wisej.NET, ciclo di vita degli oggetti dell'interfaccia, ciclo di vita di richiesta/thread e ciclo di vita della unit of work di EF Core.
- ✓ Registrare EF Core con la dependency injection di Microsoft ed esporre il service provider a Wisej.NET.
- ✓ Scegliere tra un DbContext per operazione, un DbContext per form di modifica e un confine di repository/servizio.
- ✓ Evitare istanze di DbContext statiche condivise e fughe accidentali di dati tra sessioni.

Guida al lab / esame

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



Lab pratico

Crea la soluzione Support Desk Data Console con i progetti Web, Data, Services e Tests, aggiungi i pacchetti del provider EF Core al progetto dati, crea SupportDeskContext con una factory di design time, registra AddDbContextFactory nello startup ASP.NET Core ed esponi a Wisej.NET l'IServiceProvider di Microsoft, poi risolvi un servizio di query da una Page Wisej.NET ed esegui una prima query di conteggio asincrona protetta da un flag di caricamento e da un messaggio di errore, senza ancora collegare nulla.

Punto di partenza

Apri Visual Studio e crea una nuova soluzione chiamata `SupportDesk` con un'applicazione web Wisej.NET (`SupportDesk.Web`), le librerie di classi vuote `SupportDesk.Data` e `SupportDesk.Services` e un progetto di test `SupportDesk.Tests` ; un database locale SQL Server LocalDB o SQLite è l'unico altro prerequisito. Il template web ti dà un `Program.cs` e una `Page` principale vuota; non esistono ancora entità, Context né servizi, e il modello completo arriva nel Modulo 2.

Approccio suggerito

- ✓ Aggiungi a `SupportDesk.Data` il pacchetto del provider EF Core (SQL Server o SQLite) e il pacchetto di design, poi crea `SupportDeskContext` con un'entità `Ticket` minima e `DbSet<Ticket> Tickets` , così la query di conteggio ha qualcosa da contare.
- ✓ Aggiungi `SupportDeskDesignTimeFactory` che implementa `IDesignTimeDbContextFactory<SupportDeskContext>` , così gli strumenti `dotnet ef` possono costruire il Context senza avviare l'applicazione web.
- ✓ In `Program.cs` leggi la stringa di connessione dalla configurazione, chiama `AddDbContextFactory<SupportDeskContext>` con la diagnostica riservata allo sviluppo, registra `TicketQueryService` e, dopo `Build()` , chiama `Application.Services.AddService<IServiceProvider>(app.Services)` .
- ✓ Scrivi `TicketQueryService` in `SupportDesk.Services` con una dipendenza nel costruttore da `IDbContextFactory<SupportDeskContext>` e un metodo `CountTicketsAsync` che crea, usa ed elimina un solo Context.
- ✓ In `TicketBrowserPage` risolvi il servizio con una proprietà `[Inject]` , poi collega `countButton_Click` come gestore async con un flag `_loading` , un pulsante disabilitato, una label di stato, un try / catch che mostra un messaggio comprensibile e un blocco finally che ripristina il pulsante.

Deliverable richiesti

- ✓ Soluzione con i progetti Web, Data, Services e Tests e i pacchetti del provider EF Core
- ✓ SupportDeskContext registrato tramite AddDbContextFactory con una factory di design time
- ✓ IServiceProvider di Microsoft esposto a Wisej.NET e un servizio di query risolto da una Page

- ✓ Prima query di conteggio asincrona con una protezione di caricamento e un messaggio di errore
- ✓ Nessun DbContext statico in tutta la soluzione

Domande di revisione per lo studente

- ✓ Quale ciclo di vita hai scelto per il DbContext dietro `CountTicketsAsync`, e che cosa andrebbe storto se la pagina tenesse vivo quel Context tra un click e l'altro?
- ✓ Due utenti aprono la console Support Desk nello stesso momento ed entrambi cliccano Count. Quali oggetti sono separati per sessione, quali sono condivisi, e perché questa divisione è sicura?
- ✓ Se il database non è raggiungibile, che cosa vede l'utente, che cosa fa il pulsante, e come torna a false il flag di caricamento?



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.