

Sistemi in tempo reale, pipeline in background, notifiche e importazioni

Guida della lezione · Guida al lab / esame

 LearnWisej

Guida della lezione

Job in background, avanzamento, notifiche, annullamento, code, server push. Che cosa significa per un team Wisej.NET avanzato — e come affrontare questo modulo.



Focus di produzione

Job in background, avanzamento, notifiche, annullamento, code, server push.

1. Il problema in produzione

I sistemi Wisej.NET avanzati eseguono spesso importazioni, generazione di report, integrazioni e notifiche mentre gli utenti continuano a lavorare. Queste pipeline devono essere osservabili, annullabili e sicure tra le sessioni.

2. Concetti chiave

Concetto	Perché è importante
modello dei job	Un record persistente che descrive un'operazione in background: il tipo, l'input, lo stato, l'avanzamento, il proprietario e il risultato, indipendentemente da qualsiasi schermata aperta.
coda in background	Una coda durevole che accetta i job e li elabora fuori dal ciclo delle richieste, così il lavoro di lunga durata non blocca mai l'interfaccia utente.
Application.StartTask	L'API di Wisej.NET per eseguire lavoro su un thread in background che resta collegato al contesto di sessione, così alla fine può aggiornare la UI in sicurezza.
Application.Update	Invia subito al browser le modifiche alla UI in sospeso fatte da un thread in background: è così che avanzamento e notifiche compaiono senza un'azione dell'utente.
servizio di notifica	Un servizio centrale che consegna i risultati dei job e gli avvisi alla sessione utente giusta, che la schermata di origine sia ancora aperta o no.
policy di retry	Regole su quante volte e con quale rapidità si ritenta un job fallito, con backoff, così gli errori transitori si risolvono senza martellare i sistemi a valle.
idempotenza	Progettare i job in modo che eseguirli due volte produca lo stesso risultato di eseguirli una volta, il che rende sicuri i retry e i messaggi duplicati.
audit dei job	Una registrazione di chi ha avviato ogni job, quando è stato eseguito e che cosa ha modificato, così l'elaborazione in background è tracciabile quanto quello interattivo.

3. Il filo della lezione

I job sopravvivono alle schermate

Un'importazione di lunga durata non dovrebbe dipendere dal fatto che un singolo gestore di click resti in vita. La UI avvia od osserva un job; un servizio lo esegue; un archivio dei job ne registra lo stato; i servizi di notifica pubblicano gli aggiornamenti verso le sessioni interessate.

Invia stato significativo

Gli aggiornamenti in tempo reale dovrebbero comunicare tappe significative: in coda, avviato, validato, importato, fallito, completato. Chiamare update di continuo spreca risorse e rende più difficile il debug.

Separare l'esecuzione dalla visualizzazione

L'Import Center del lab ha due metà che non condividono mai un oggetto. Un servizio dei job esegue le importazioni: prende una definizione di job, legge il file, valida e scrive le righe a blocchi, e scrive avanzamento e risultato in un archivio dei job. La schermata osserva: avvia un job chiamando il servizio, poi mostra ciò che l'archivio dice di esso. La schermata può essere chiusa, riaperta in una nuova sessione o aperta da un altro utente con il permesso adeguato, e il job non lo sa né se ne cura.

Il modello dei job è il contratto tra le due metà. Ogni record di job contiene un identificativo, il tipo di lavoro, chi l'ha avviato e in quale tenant, lo stato, un valore di avanzamento, un messaggio, i timestamp e un riepilogo del risultato con gli errori per riga. Poiché il record è persistente, lo stato sopravvive a un riavvio del server, e un tecnico del supporto può vedere tutti i job della giornata senza chiedere a nessuno.

La coda in background sta tra le due metà. Avviare un job lo mette in coda; un worker lo prende in carico. È questo che rende il thread di esecuzione indipendente dalla richiesta che l'ha chiesto.

Aggiornamenti in tempo reale con intenzione

Wisej.NET può inviare modifiche al browser senza una richiesta, ed è così che l'avanzamento compare mentre l'utente aspetta. Un task in background avviato con `Application.StartTask` mantiene il contesto di sessione, modifica la barra di avanzamento e la label di stato e chiama `Application.Update` per inviare quelle modifiche al browser. Usato con leggerezza, lo stesso meccanismo invia un aggiornamento per ogni riga importata e trasforma un'importazione di due minuti in una valanga di round trip che rallenta il server e il browser.

Il `ImportWorkOrdersJob` del lab pubblica invece delle tappe: in coda, avviato, validato, ogni blocco completato, fallito, completato. Un milestone sink le riceve e decide se aggiornare la UI subito o aspettare, così un job che produce mille eventi genera una manciata di aggiornamenti. Ogni aggiornamento porta uno stato significativo, che è ciò che l'utente vuole davvero sapere.

La seconda domanda del lab, con quale frequenza si aggiorna la UI, in una buona progettazione ha una risposta numerica, ed è al massimo un piccolo numero al secondo.

Modelli di stato dei job e di notifica

Lo stato di un job è una piccola macchina a stati: in coda, in esecuzione, completato, fallito, annullato. L'avanzamento è un valore separato, perché un job in esecuzione al novanta per cento e uno al dieci per cento hanno lo stesso stato. L'archivio degli stati espone il record corrente e uno storico, così la schermata di dettaglio del job può mostrare quando è avvenuta ogni transizione e quale messaggio l'ha accompagnata.

Le notifiche sono il modo in cui il risultato raggiunge una persona che non sta più guardando. Un servizio di notifica accetta un messaggio per un utente o un ruolo, lo memorizza e lo consegna a qualsiasi sessione attiva di quell'utente tramite la campanella dell'Import Center del lab. Se l'utente non ha fatto login, la notifica aspetta finché non lo fa. Anche le notifiche sono record, con uno stato letto e non letto, così nulla va perso quando un browser si chiude nel momento sbagliato.

Entrambi i modelli sono consapevoli del tenant. Un job appartiene a un tenant, le sue notifiche appartengono a utenti di quel tenant, e l'archivio non restituisce mai un job oltre il confine del tenant.

Annullamento, retry e sessioni chiuse

Un pulsante di annullamento imposta un flag; da solo non ferma nulla. Il job rispetta l'annullamento controllando il flag tra un blocco e l'altro, terminando in modo pulito il blocco corrente, registrando ciò che è stato fatto e passando allo stato annullato. Un annullamento che si ferma a metà riga lascia dati scritti a metà, il che è peggio che finire il blocco. La policy di retry distingue gli errori transitori, come un timeout nella scrittura di una riga, che vengono ritentati un numero limitato di volte con un ritardo, dagli errori definitivi, come un file malformato, che fanno fallire subito il job.

Il fallimento parziale viene riportato per riga. Un'importazione di diecimila righe di cui dodici errate si completa con un risultato che elenca le dodici, e l'utente corregge e reimporta quelle, non l'intero file. L'elaborazione idempotente delle righe lo rende sicuro: importare due volte la stessa riga produce un solo record.

La chiusura della sessione è il percorso di errore del lab. L'utente chiude il browser a importazione in corso, il job continua a girare perché appartiene al servizio e non alla sessione, e alla riapertura la schermata trova il job nell'archivio e ne mostra lo stato finale. La domanda su a chi appartiene il job dopo la chiusura della sessione ha una sola risposta: al servizio.



La regola di produzione avanzata

Non fermarti quando la schermata sembra corretta. Prima di considerare completo il modulo, spiega a chi appartiene lo stato, le implicazioni di sicurezza, i percorsi di errore e il comportamento in produzione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Progettare pipeline in background che separano l'esecuzione dei job dalla visualizzazione dell'avanzamento nella UI.
- ✓ Usare gli aggiornamenti in tempo reale in modo intenzionale ed evitare aggiornamenti eccessivi al client.
- ✓ Implementare modelli di notifica e di stato dei job.
- ✓ Gestire annullamento, retry, fallimenti parziali e chiusura della sessione.

Guida al lab / esame

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



Lab pratico

Costruisci un Import Center con una coda di job, un osservatore dell'avanzamento, annullamento, elaborazione delle righe con retry, notifiche e una schermata di dettaglio del job. Dimostra il ripristino dopo aver chiuso e riaperto la sessione.

Approccio suggerito

- ✓ Leggi il problema in produzione e individua il rischio architetturale prima di toccare la UI.
- ✓ Crea o aggiorna per primo l'incapsulamento in un servizio, comando, modello o componente coinvolto.
- ✓ Costruisci la schermata o il componente Wisej.NET seguendo gli standard di progetto stabiliti.
- ✓ Aggiungi una dimostrazione del percorso di errore, non solo del percorso felice.
- ✓ Annota che cosa è cambiato nel registro di apprendimento e aggiorna la checklist relativa.

Deliverable richiesti

- ✓ Modello dei job e archivio degli stati
- ✓ Astrazione della coda in background
- ✓ UI dell'osservatore dell'avanzamento
- ✓ Pannello delle notifiche
- ✓ Policy di retry e di annullamento

Domande di revisione per lo studente

- ✓ A chi appartiene il job dopo la chiusura della sessione?
- ✓ Con quale frequenza si aggiorna la UI?
- ✓ Quali errori consentono un retry e quali sono definitivi?



La regola di produzione avanzata

Non fermarti quando la schermata sembra corretta. Prima di considerare completo il modulo, spiega a chi appartiene lo stato, le implicazioni di sicurezza, i percorsi di errore e il comportamento in produzione.