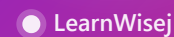


Base de datos, E/S de archivos, async y esperas externas

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

La latencia que CPU Usage no puede ver: la herramienta Database para los tiempos de consulta y los patrones N+1, File I/O para el trabajo de importación y exportación, y .NET Async para las cadenas bloqueadas o secuenciales por accidente. Qué significa esto para un desarrollador de Wisej.NET que diagnostica cuellos de botella en una aplicación en producción, y cómo leer este módulo.



Enfoque en producción

La latencia que CPU Usage no puede ver: la herramienta Database para los tiempos de consulta y los patrones N+1, File I/O para el trabajo de importación y exportación, y .NET Async para las cadenas bloqueadas o secuenciales por accidente.

1. El problema en producción

El Módulo 2 midió 1840 ms de tiempo transcurrido y solo encontró 300 ms de CPU. El Módulo 3 eliminó la mayor parte de esa CPU y la pantalla sigue sin ser rápida, porque los 600 ms que faltan nunca fueron cálculo: eran espera. Esta es la categoría peor diagnosticada en una aplicación de gestión: el servidor está inactivo, el gráfico de CPU está plano y el usuario mira un indicador de carga. En algún sitio hay una consulta que se ejecuta una vez por fila, un formulario de detalle que trae una columna que nadie muestra, o un controlador de clic que se bloquea en `.Result` mientras un hilo del thread pool no hace nada. Las herramientas que ven esto son Database, File I/O, .NET Async e Instrumentation, y en este módulo la latencia que le queda a WisejPerfLab por fin tiene nombre.

2. Conceptos clave

Concepto	Por qué importa
herramienta Database	Registra las consultas de ADO.NET y EF Core que lanza un escenario, con su duración y su orden, para que cada consulta pueda asociarse a la acción de interfaz que la provocó.
consulta N+1	Una consulta para listar y luego una más por cada elemento devuelto; la lista va bien con diez filas y se hunde con mil, y es invisible para el profiling de CPU.
traer datos de más	Seleccionar columnas o filas que la pantalla nunca muestra; las grandes columnas de texto y binarias son las culpables habituales, y cuestan tiempo de consulta, transferencia y memoria.
consulta sin seguimiento	Una consulta de solo lectura que no prepara el seguimiento de cambios, adecuada para grids y listas que muestran datos en lugar de editarlos.
herramienta .NET Async	Muestra las actividades asíncronas y dónde esperan, y así se hace visible una cadena secuencial por accidente o bloqueada.
bloquear sobre código async	Llamar a <code>.Result</code> o <code>.Wait()</code> desde un controlador; retiene un hilo mientras espera y, bajo carga, produce síntomas que parecen agotamiento del thread pool.
exportación en segundo plano	La generación larga de archivos o informes sacada de la ruta inmediata del clic con <code>Application.StartTask</code> , informando del progreso a intervalos significativos y no en cada paso.

3. Desarrollo de la lección

Asociar consultas a clics

Ejecuta un escenario con la herramienta Database y lee la lista de consultas como una historia. La carga de un grid no debería lanzar una consulta por cada fila mostrada. Un editor de consulta no debería recargar datos de referencia sin cambios en cada interacción. Un formulario de detalle no debería seleccionar una gran columna de descripción o de adjuntos cuando la pantalla muestra un resumen. Cada una de esas situaciones es evidente en cuanto las consultas aparecen junto a la acción que las produjo, y casi imposible de detectar en una revisión de código.

WisejPerfLab tiene dos de las tres. El grid de tickets lanza una consulta de cliente por fila, así que la búsqueda que devuelve doscientas filas lanza doscientas una consultas; y la consulta de detalles selecciona todas las columnas del ticket, incluido el texto completo de la conversación. La corrección combinada es una única proyección sin seguimiento que selecciona exactamente las columnas del grid, une el nombre del cliente en lugar de cargarlo de forma diferida, y ordena y pagina el resultado. Los índices importan para el filtro y el join, y las split queries son una herramienta contra la explosión cartesiana, no algo que espolvorear por defecto sobre todas las consultas.

Async va de hilos, no de velocidad

El segundo escenario es la exportación CSV, y falla de dos maneras distintas a la vez. El controlador del clic llama a un servicio async y se bloquea en `.Result`, lo que retiene un hilo durante toda la generación; y el

escritor emite muchas escrituras pequeñas en lugar de hacer streaming. Con un solo usuario ninguna de las dos es fatal. Con cincuenta sesiones concurrentes, los hilos bloqueados se acumulan y los síntomas parecen otra cosa completamente distinta: toda la aplicación se ralentiza, incluidas pantallas que no tienen nada que ver con las exportaciones.

Merece la pena ser preciso sobre lo que hace `async`. Esperar con `await` una operación de E/S libera el hilo mientras dura la espera, lo que mejora cuántas peticiones concurrentes puede soportar un servidor. No hace que la operación termine antes ni reduce el trabajo de CPU. Así que `async` es la respuesta correcta a «tengo todos los hilos bloqueados» y la respuesta equivocada a «este método es lento». Dos advertencias más: nunca ejecutes operaciones en paralelo sobre objetos que no sean seguros para hilos, y mantén las actualizaciones de interfaz en el contexto de la sesión en lugar de lanzarlas desde hilos arbitrarios.

```
public async Task<IReadOnlyList<TicketGridRow>> SearchTicketsAsync(TicketFilter f,
    CancellationToken ct)
    => await _db.Tickets
        .AsNoTracking() // read-only list: no change tracking
        .Where(t => t.Status == f.Status)
        .OrderByDescending(t => t.UpdatedAt)
        .Select(t => new TicketGridRow // projection, not the entity
        {
            Id = t.Id, Number = t.Number,
            Customer = t.Customer.Name, // joined, not lazy-loaded per row
            Status = t.Status, UpdatedAt = t.UpdatedAt
        })
        .Take(f.PageSize) // the page, not the table
        .ToListAsync(ct);
```

Sacar la exportación de la ruta del clic

El profiling de File I/O muestra la otra mitad: lecturas y escrituras pequeñas repetidas donde bastaría un stream, y una plantilla que se vuelve a leer en cada petición. El patrón que lo corrige es el que el curso ha ido construyendo. Desactiva el botón, inicia el trabajo con `Application.StartTask`, escribe la salida en streaming en lugar de montarla en memoria, y envía el progreso en pasos significativos: cada diez por ciento, no por cada fila. Reactiva el botón y entrega el archivo en un `finally`, y comprueba `IsDisposed` antes de tocar los controles, porque el usuario puede haberse ido a otra parte.

La caché merece aquí una línea de criterio. Las plantillas de exportación son compartidas e inmutables, así que guardarlas en caché es capacidad gratuita. Los documentos generados son de cada usuario y a menudo confidenciales, así que guardarlos en caché globalmente es a la vez un problema de memoria y un riesgo de divulgación. La distinción es la misma que trazaron el Módulo 3 y el Módulo 4, y no cambia para los archivos.

Demostrar que se eliminó una espera

La evidencia de este módulo no es una cifra de CPU. Para la búsqueda, informa del número de consultas antes y después, de la duración de la consulta más lenta y del tiempo transcurrido del escenario según la sonda que está en su sitio desde el Módulo 1. Para la exportación, informa del tiempo de reloj, de si el controlador sigue bloqueando un hilo y del patrón de E/S de la traza de File I/O. Después, si puedes, vuelve a ejecutarlo con un número realista de sesiones concurrentes: los problemas de espera son la

categoría que peor se comporta con concurrencia, y una corrección que solo aguanta con un usuario no es una corrección.



Errores habituales

Tomar una CPU baja como prueba de que la aplicación es rápida. Usar `.Result` en un controlador de clic y provocar síntomas de agotamiento del thread pool. Permitir que la carga de un grid produzca una consulta por cada fila mostrada. Recargar datos de consulta estáticos en cada acción sin medir. Guardar en caché globalmente documentos generados privados de cada usuario.



Regla de evidencia

Define el escenario antes de abrir un profiler, recoge desde una compilación Release sin el depurador, cambia una sola cosa y vuelve a ejecutar el mismo escenario para comparar. Clasifica el retraso en uno de los cinco grupos —CPU en tu código, bloqueos y esperas, asignación y retención, acceso a datos y E/S, o carga útil de actualización—, porque cada grupo tiene su propia herramienta de Visual Studio, y una optimización que no puedes defender con una traza de antes y otra de después es una conjetura.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Recoger una traza de Database de Visual Studio para las consultas de ADO.NET o EF Core que lanzan las pantallas de Wisej.NET.
- ✓ Encontrar consultas lentas, consultas repetidas, patrones N+1 y consultas que traen datos de más.
- ✓ Usar .NET Async e Instrumentation para identificar flujos async bloqueados o secuenciales.
- ✓ Perfilar escenarios de importación/exportación de archivos y sacar la E/S costosa de los eventos inmediatos de la interfaz.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Perfila los dos escenarios dominados por esperas en WisejPerfLab. Ejecuta la búsqueda de tickets con la herramienta Database y asocia cada consulta a la acción de interfaz: la carga del grid lanza una consulta de cliente por cada fila mostrada y la consulta de detalles trae grandes columnas de texto que nadie muestra. Sustituye ambas por una única consulta de proyección sin seguimiento que seleccione solo las columnas del grid y pague el resultado. Después ejecuta la exportación CSV con File I/O y .NET Async, encuentra la llamada bloqueante .Result en el controlador del clic y las escrituras pequeñas repetidas, y traslada la generación a un flujo en segundo plano con Application.StartTask que escriba el archivo en streaming e informe del progreso de forma limitada en lugar de hacer push en cada paso.

Punto de partida

WisejPerfLab tras el Módulo 5: el grid es virtual y usa una proyección para mostrar los datos, pero la consulta que hay detrás sigue cargando el cliente de forma diferida por cada fila, y el formulario de detalle sigue seleccionando todas las columnas, incluido el texto de la conversación. El botón de exportación CSV llama a `_exportService.BuildAsync( ... ).Result` y escribe el archivo línea a línea. La sonda del Módulo 1 sigue midiendo los tres escenarios.

Enfoque sugerido

- ✓ Recoge una traza de Database de la búsqueda de tickets en Release y enumera las consultas en orden junto a las acciones de interfaz que las produjeron. Cuéntalas, y anota la duración de la más lenta y las columnas que selecciona la consulta de detalles.
- ✓ Sustituye la búsqueda de cliente por fila y la consulta de detalles que trae datos de más por una única proyección `AsNoTracking` que seleccione exactamente las columnas del grid, una el nombre del cliente, ordene y pague. Comprueba que las columnas del filtro y del join están indexadas antes de concluir que la consulta en sí es lenta.
- ✓ Recoge una traza de .NET Async de la exportación, y una de Instrumentation si la espera sigue sin estar clara, para demostrar dónde se bloquea el controlador del clic. Confirma que la llamada `.Result` retiene un hilo y no es simplemente poco elegante.
- ✓ Traslada la generación de la exportación a `Application.StartTask`: desactiva el botón, escribe la salida en streaming en lugar de montarla en memoria, envía el progreso a intervalos significativos y no por fila, y reactiva el botón y entrega el archivo en un `finally` tras comprobar `IsDisposed`.
- ✓ Vuelve a ejecutar ambos escenarios y, si puedes, repite la exportación con varias sesiones concurrentes: los problemas de espera se comportan peor con concurrencia, y una mejora con un solo usuario puede no sobrevivir a ella.

Entregables requeridos

- ✓ Traza de Database que asocia las consultas a las acciones de interfaz, con el número de consultas N+1 antes de la corrección
- ✓ Única consulta de proyección sin seguimiento con paginación que sustituye las consultas por fila y las que traen datos de más
- ✓ Evidencia de File I/O y .NET Async para la exportación, que nombra la llamada bloqueante y el patrón de escritura
- ✓ Exportación trasladada a una tarea en segundo plano que escribe la salida en streaming y hace push del progreso solo en los pasos significativos
- ✓ Número de consultas, duración de las consultas y tiempo de reloj de la exportación antes/después, con los riesgos restantes

Preguntas de repaso

- ✓ La búsqueda lanza ahora una consulta en lugar de doscientas una, pero el tiempo transcurrido solo se ha reducido a la mitad. ¿Dónde está el resto del tiempo y qué herramienta usarías para averiguarlo?
- ✓ Un compañero dice que la exportación está bien porque `.Result` «solo bloquea un hilo». Explica qué ocurre con cincuenta exportaciones concurrentes y qué pantallas notaría el usuario que se ralentizan.
- ✓ De lo que este laboratorio guarda en caché, ¿qué es seguro compartir entre sesiones y qué no, y qué regla usaste para decidirlo?



Regla de evidencia

Define el escenario antes de abrir un profiler, recoge desde una compilación Release sin el depurador, cambia una sola cosa y vuelve a ejecutar el mismo escenario para comparar. Clasifica el retraso en uno de los cinco grupos —CPU en tu código, bloqueos y esperas, asignación y retención, acceso a datos y E/S, o carga útil de actualización—, porque cada grupo tiene su propia herramienta de Visual Studio, y una optimización que no puedes defender con una traza de antes y otra de después es una conjetura.