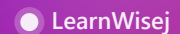


Dibujo de celdas en DataGridView

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Dibujar dentro de la cuadrícula: UserPaint en el ámbito de celda o de columna, DataGridViewCellPaintEventArgs con RowIndex y ColumnIndex, celdas de progreso y sparkline que leen las filas de forma barata, y la comparación honesta con el contenido de celda AllowHtml en una cuadrícula de mil filas. Qué significa esto para un desarrollador .NET con experiencia que ya construye pantallas Wisej.NET y ahora necesita gráficos personalizados, y cómo leer este módulo.



Enfoque de producción

Dibujar dentro de la cuadrícula: UserPaint en el ámbito de celda o de columna, DataGridViewCellPaintEventArgs con RowIndex y ColumnIndex, celdas de progreso y sparkline que leen las filas de forma barata, y la comparación honesta con el contenido de celda AllowHtml en una cuadrícula de mil filas.

1. El problema en producción

El Módulo 2 dio a VisualOperationsStudio un **TelemetryGauge** : un control, una máquina, unos cientos de redibujados por hora. La cuadrícula de operaciones es otra cosa. Un supervisor de turno abre una lista de mil máquinas y quiere ver de un vistazo cuáles están cerca de su límite y cuáles llevan toda la mañana desviándose. Unos números en una columna no consiguen eso, y un indicador por fila serían mil controles del lado del servidor en una sola sesión. Es la propia cuadrícula la que tiene que dibujar, una vez por celda visible, y parar en cuanto la celda se desplaza fuera de la vista.

2. Conceptos clave

Concepto	Por qué importa
<code>UserPaint</code>	El interruptor explícito: <code>DataGridViewCell.UserPaint = true</code> para una celda, <code>DataGridViewColumn.UserPaint = true</code> para una columna entera. Sin él, la cuadrícula nunca te pide que dibujes.
<code>CellPaint</code>	El evento a nivel de cuadrícula que se dispara para las celdas dibujadas por el usuario cuando hay que dibujarlas; te suscribes una vez en la cuadrícula, no una vez por fila, y distribuyes dentro del controlador.
<code>DataGridViewCellPaintEventArgs</code>	Deriva de <code>PaintEventArgs</code> , así que lleva <code>Graphics</code> y <code>ClipRectangle</code> , y añade <code>RowIndex</code> y <code>ColumnIndex</code> para que un solo controlador pueda atender varias columnas.
sin ayudantes de WinForms	Los argumentos de evento documentados de Wisej.NET no exponen los métodos de conveniencia de WinForms <code>PaintBackground()</code> y <code>PaintContent()</code> . Tu controlador es dueño de todo el resultado visual de esa celda.
<code>e.ClipRectangle</code> como superficie	El rectángulo de destino local de la celda. Las columnas se pueden redimensionar y la cuadrícula puede disponerse de forma adaptable, así que cada coordenada que dibujes tiene que derivarse de él y no de una constante.
ayudantes de dibujo puros	Métodos con la forma <code>DrawProgressCell(Graphics g, Rectangle rect, double value)</code> , para que el controlador se convierta en código de distribución y ese mismo ayudante pueda dibujar más adelante en un mapa de bits fuera de pantalla.
contenido de celda <code>AllowHtml</code>	La alternativa nativa del navegador: iconos, spans, etiquetas y CSS representados por el cliente, y la razón por la que la documentación reserva <code>CellPaint</code> para elementos visuales demasiado complejos para HTML.

3. Desarrollo de la lección

Convertir una celda en una superficie de dibujo

En un `DataGridView` de Wisej.NET nada se dibuja por el usuario de forma predeterminada. Te acoges a ello celda a celda o columna a columna poniendo `UserPaint` en `true`, y solo entonces dispara la cuadrícula el `CellPaint` de esa superficie. Olvidar el interruptor es el motivo más habitual de que un controlador de aspecto correcto no llegue a ejecutarse nunca, así que mantén juntas la línea que lo activa y la de la suscripción en tu método de configuración de columnas.

Prefiere el ámbito de columna. En el laboratorio añades columnas explícitas a `grid0operations` y activas `UserPaint = true` solo en las columnas Health y Trend; después te suscribes a `CellPaint` una vez en la cuadrícula. Eso te da una suscripción para toda la pantalla en lugar de una por fila, y mantiene todas las demás columnas —identificador, nombre de máquina, planta, última lectura— como contenido corriente

de cuadrícula que el navegador puede seleccionar, buscar y escalar. Activar el dibujo por el usuario para toda la cuadrícula es la decisión contraria: te entrega celdas que nunca quisiste dibujar y obliga al controlador a filtrarlas en cada redibujado.

Qué te entrega el evento y qué no

Como `DataGridViewCellPaintEventArgs` deriva de `PaintEventArgs`, la parte de dibujo es lo que enseñó el Módulo 2: `e.Graphics` es la superficie, `e.ClipRectangle` es tu área, y liberas todos los pens, brushes y paths que crees mientras dejas en paz `e.Graphics`. Lo nuevo es la distribución. Tanto `e.RowIndex` como `e.ColumnIndex` necesitan una comprobación antes de tocar nada: un `RowIndex` negativo no es una fila de datos, y una columna a la que no te acogiste debe salir de inmediato.

La parte reducida de la API es la que suele pillar a la gente. Los argumentos de evento de Wisej no exponen `PaintBackground()` ni `PaintContent()`, así que un ejemplo de WinForms que los llame hay que reescribirlo en lugar de compilarlo, y decides tú mismo qué aspecto tienen el fondo y el contenido de la celda. Tómalo como una ventaja: cuando el controlador produce la celda entera, no hay ningún estado a medio dibujar del que preocuparse.

```
private void gridOperations_CellPaint(object sender, DataGridViewCellPaintEventArgs e)
{
    if (e.RowIndex < 0)
        return;

    var rect = Rectangle.Inflate(e.ClipRectangle, -4, -5);

    if (e.ColumnIndex == healthColumn.Index)
        DrawHealthBar(e.Graphics, rect, Convert.ToInt32(
            gridOperations.GetValue(e.ColumnIndex, e.RowIndex) ?? 0));
    else if (e.ColumnIndex == trendColumn.Index)
        DrawSparkline(e.Graphics, rect, TrendOf(e.RowIndex));
}
```

Celdas de progreso, de estado y de sparkline

Tres formas cubren la mayoría de las cuadrículas de operaciones. Una celda de progreso rellena una pista inactiva con un rectángulo activo cuyo ancho es proporcional a un valor normalizado, y dibuja un número solo cuando la columna es lo bastante ancha. Una celda de estado es una sola elipse rellena, a veces con una etiqueta corta; es el elemento visual útil más barato de la cuadrícula y a menudo más claro que un icono. Una sparkline normaliza una serie corta al rectángulo de la celda y une los puntos con `DrawLines`, como hace la columna Trend en el laboratorio: su trabajo es resumir, nunca reproducir una serie de diez mil puntos dentro de cuarenta píxeles.

Mantén los cálculos fuera del controlador. Dale a cada elemento visual un ayudante que reciba un `Graphics`, un `Rectangle` y los valores que necesite, y el controlador se reducirá a una comprobación y dos llamadas. El renderizador fuera de pantalla del Módulo 6 podrá entonces llamar a ese mismo método para poner la sparkline idéntica en un PNG exportado. No olvides los estados que gestiona la cuadrícula: una celda dibujada por el usuario tiene que seguir siendo legible cuando su fila está seleccionada, así que elige colores conscientes de la selección o mantén el elemento visual sobrio y deja que hable el resaltado

de la fila. No dejes nunca que el color sea el único portador de un estado crítico; ese mismo valor debería existir también como texto o como tooltip.

Cuándo dibujar es la herramienta equivocada

Wisej.NET también representa HTML dentro de las celdas mediante `AllowHtml`, y la documentación sitúa `CellPaint` como la respuesta para contenido demasiado complejo para HTML o para código de dibujo .NET existente que quieras reutilizar. Elige HTML cuando el elemento visual se pueda expresar con texto, iconos, spans y CSS, cuando la búsqueda, la selección, la accesibilidad o el escalado del navegador se beneficien de contenido real del DOM, o cuando haya muchos miles de celdas. Elige el dibujo cuando necesites geometría a nivel de píxel, cuando el elemento visual sea compacto y cambie poco, o cuando el HTML acabaría siendo más complicado que el código de dibujo.

Una segunda alternativa no tiene nada que ver con HTML: los propios tipos de celda de la cuadrícula y su capacidad de alojar controles. Un botón, una casilla de verificación, un selector de fecha o un cuadro combinado deben seguir siendo celdas interactivas reales. Dibujar es un modelo de representación, no de interacción, y un botón dibujado te obliga a reconstruir a mano la respuesta al puntero, el acceso por teclado, el foco y los comandos. Por eso el laboratorio construye el indicador de estado dos veces y te pide que dejes por escrito cuál de los dos llevarías a producción.

Desplazarse por mil filas

Un `DataGridView` gestiona bien los conjuntos de datos grandes con enlace a datos y modos virtuales, y un controlador de `CellPaint` descuidado deshace esa ventaja en una sola línea. El desplazamiento vuelve a llamar al controlador por cada celda que se hace visible, así que cualquier operación cara se multiplica por lo rápido que el usuario arrastre la barra de desplazamiento.

- No hagas nunca E/S, ni llamadas a base de datos o a servicios, dentro del controlador.
- No cargues imágenes ni crees fuentes por celda; constrúyelas una vez y reutilízalas.
- Dibuja el menor conjunto posible de columnas y reduce el número de puntos de las sparklines.
- Mantén la obtención del estado en $O(1)$: indexa la lista que enlazaste, o lee el valor de la celda a través de la cuadrícula, pero nunca busques en una colección por celda.
- Mide con un número de filas realista: cien filas no demuestran nada sobre mil.

Las mil filas generadas del laboratorio existen precisamente por ese último punto. Enlázalas, arrastra la barra de desplazamiento de arriba abajo y compara lo que cuestan las dos columnas dibujadas frente a la columna HTML que tienen al lado. Esa medición, y no una preferencia, es lo que debe figurar en tus notas.



Regla de dibujo: primero la superficie

Elige la superficie de renderizado antes de escribir una sola llamada de dibujo. Pregúntate dónde se producen los píxeles y qué cruza la red: un controlador de Paint de un control se ejecuta en el servidor y envía una imagen, CellPaint hace lo mismo dentro de una celda de cuadrícula, Canvas envía comandos de dibujo que ejecuta el navegador y no conserva nada tras un redimensionado, y System.Drawing.Managed produce bytes de imagen sin ningún control de por medio. Mantén el modelo semántico en objetos C#, haz que cada renderizador sea una función determinista de ese modelo, invalida solo cuando haya cambiado una dependencia visual, y deja siempre un equivalente textual de todo lo importante que exista únicamente como píxeles.



Objetivos de aprendizaje

Al final de este módulo podrás:

- ✓ Activar el dibujo en el ámbito de celda o de columna con UserPaint.
- ✓ Usar DataGridViewCellPaintEventArgs de forma segura.
- ✓ Construir celdas de progreso, de estado y de sparkline.
- ✓ Comparar el dibujo por el usuario con el HTML de celda, los tipos de celda integrados, los controles en celda y las imágenes.
- ✓ Diseñar el dibujo de celdas para conjuntos de datos grandes sin convertir el desplazamiento en un cuello de botella de renderizado.

Guía del laboratorio / examen

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



Laboratorio práctico

Dota a VisualOperationsStudio de una cuadrícula de operaciones con dos columnas dibujadas por el usuario. Añade columnas explícitas a gridOperations, activa UserPaint = true solo en las columnas Health y Trend y gestiona CellPaint una única vez: sal de inmediato cuando e.RowIndex sea negativo o e.ColumnIndex no sea una de las dos columnas objetivo, y después dibuja una barra de progreso coloreada por umbrales para Health y una sparkline de polilínea para Trend dentro de e.ClipRectangle. Lee los valores de cada fila del objeto MachineStatus enlazado en lugar de buscar en una lista dentro del controlador, respeta los colores de la fila seleccionada y mantén el controlador libre de efectos secundarios. Enlaza 1000 filas generadas, desplázate por ellas y después construye el mismo indicador de estado una segunda vez como celda AllowHtml, para poder dejar por escrito cuál de los dos llevarías a producción.

Punto de partida

Abres VisualOperationsStudio tal como lo dejó el Módulo 2: la `VisualOperationsPage` lleva el control `TelemetryGauge` y su ayudante `GaugeGeometry`, y la página tiene sitio para una cuadrícula pero aún no tiene datos. Añade una clase `MachineStatus` con un identificador, un nombre, una planta, un porcentaje `Health` y una lista corta de lecturas recientes para la tendencia, más un generador capaz de producir mil de ellas.

Enfoque sugerido

- ✓ Añade `gridOperations` con columnas explícitas para identificador, máquina, planta, Health y Trend, enlaza primero diez filas `MachineStatus` y confirma que las columnas simples se representan antes de que exista ningún dibujo.
- ✓ Activa `UserPaint = true` solo en las columnas Health y Trend, suscríbete a `gridOperations.CellPaint` una única vez y empieza el controlador con una comprobación que salga cuando `e.RowIndex` sea negativo o `e.ColumnIndex` no sea ninguna de las dos columnas objetivo.
- ✓ Escribe `DrawHealthBar(Graphics, Rectangle, int)` y `DrawSparkline(Graphics, Rectangle, IReadOnlyList<double>)` como ayudantes puros que deriven todas las coordenadas del rectángulo que reciben, elijan el color de relleno a partir de los umbrales de advertencia y crítico, y liberen cada pen y cada brush que creen.
- ✓ Convierte el controlador en pura distribución: reduce `e.ClipRectangle` con `Rectangle.Inflate`, obtén los valores de la fila en un solo paso O(1), pásalos al ayudante y mantén legible la fila seleccionada eligiendo colores que sigan contrastando con el fondo de la selección.
- ✓ Enlaza 1000 filas generadas, arrastra la barra de desplazamiento de arriba abajo, después añade una tercera columna de estado que use `AllowHtml` para representar el mismo estado de advertencia o crítico, vuelve a desplazarte y registra ambos resultados.

Entregables requeridos

- ✓ gridOperations con columnas explícitas y UserPaint = true solo en las columnas Health y Trend
- ✓ Un único controlador de CellPaint que comprueba e.RowIndex y e.ColumnIndex antes de dibujar nada
- ✓ Celda de progreso coloreada por umbrales y celda de sparkline de polilínea dibujadas dentro de e.ClipRectangle
- ✓ Valores de fila leídos del objeto MachineStatus enlazado, sin búsquedas, reservas de memoria ni creación de controles por celda
- ✓ Una columna de estado con AllowHtml construida en paralelo, más una nota que compare ambas en un desplazamiento por 1000 filas

Preguntas de repaso

- ✓ Un compañero pega un ejemplo de dibujo de celdas de WinForms que empieza llamando a `e.PaintBackground( ... )`. Explica qué te dan en su lugar los argumentos de evento de Wisej.NET y qué tiene que decidir ahora el controlador por sí mismo.
- ✓ Tu ayudante de sparkline fija en el código un ancho de trazado de 100 píxeles y se ve bien en tu pantalla. Describe dos acciones del usuario que lo rompen y qué debería leer el ayudante en su lugar.
- ✓ La columna de estado en HTML y la columna Health dibujada muestran las dos una máquina en estado crítico. Argumenta cuál llevarías a producción en una cuadrícula que el personal de soporte busca y lee con un lector de pantalla, y qué perderías en cada caso.



Regla de dibujo: primero la superficie

Elige la superficie de renderizado antes de escribir una sola llamada de dibujo. Pregúntate dónde se producen los píxeles y qué cruza la red: un controlador de Paint de un control se ejecuta en el servidor y envía una imagen, CellPaint hace lo mismo dentro de una celda de cuadrícula, Canvas envía comandos de dibujo que ejecuta el navegador y no conserva nada tras un redimensionado, y System.Drawing.Managed produce bytes de imagen sin ningún control de por medio. Mantén el modelo semántico en objetos C#, haz que cada renderizador sea una función determinista de ese modelo, invalida solo cuando haya cambiado una dependencia visual, y deja siempre un equivalente textual de todo lo importante que exista únicamente como píxeles.