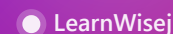


La arquitectura de dibujo de Wisej.NET

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Cuatro canalizaciones de renderizado y cómo elegir entre ellas: el Paint de un control, que envía una imagen renderizada en el servidor; el CellPaint de la DataGridView, dentro de una sola celda; Wisej.Web.Canvas, que envía comandos de dibujo al navegador; y System.Drawing.Managed fuera de pantalla, que produce bytes de imagen; además de la escala de decisión que mantiene los controles estándar y el CSS como opción por defecto. 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

Cuatro canalizaciones de renderizado y cómo elegir entre ellas: el Paint de un control, que envía una imagen renderizada en el servidor; el CellPaint de la DataGridView, dentro de una sola celda; Wisej.Web.Canvas, que envía comandos de dibujo al navegador; y System.Drawing.Managed fuera de pantalla, que produce bytes de imagen; además de la escala de decisión que mantiene los controles estándar y el CSS como opción por defecto.

1. El problema en producción

VisualOperationsStudio tiene que mostrar la misma lectura de máquina en cuatro sitios: un dial en la página de operaciones, una barra de salud dentro de una fila de cuadrícula, un nodo en un mapa de topología y un PNG adjunto al informe nocturno de turno. Todos son «dibujar un número» y, sin embargo, echar mano de la API de dibujo que recuerdes del escritorio produce cuatro fragmentos de código con comportamientos radicalmente distintos en CPU, ancho de banda y redimensionado. Este módulo pone la elección de la superficie por delante del código de dibujo, porque la superficie decide dónde se producen los píxeles y qué viaja hasta el navegador.

2. Conceptos clave

Concepto	Por qué importa
<code>Paint</code> / <code>PaintEventArgs</code>	El evento de dibujo a nivel de control: tu C# se ejecuta en el servidor sobre el <code>Graphics</code> que proporciona, dentro del <code>ClipRectangle</code> que indica, y Wisej.NET convierte el resultado en una imagen para el navegador.
<code>DataGridView.CellPaint</code>	El mismo mecanismo de dibujo acotado a una celda; su <code>DataGridViewCellPaintEventArgs</code> deriva de <code>PaintEventArgs</code> y añade <code>RowIndex</code> y <code>ColumnIndex</code> .
<code>UserPaint</code>	El interruptor que necesita una celda o una columna entera para que la cuadrícula dispare <code>CellPaint</code> , lo que mantiene el dibujo como algo opcional por columna en lugar de extenderlo a toda la cuadrícula.
<code>Wisej.Web.Canvas</code>	Una superficie 2D de modo inmediato en el navegador que refleja el modelo <code>CanvasRenderingContext2D</code> de HTML5; las llamadas de dibujo las ejecuta el cliente, no se renderizan como imagen en el servidor.
<code>Redraw</code>	El evento del Canvas en el que se reconstruye toda la escena a partir de tu modelo, porque el canvas guarda píxeles y los pierde cuando se recrea o se redimensiona.
<code>System.Drawing.Managed</code>	Renderizado fuera de pantalla sin ningún control visible: crea un <code>Bitmap</code> , obtén su <code>Graphics</code> con <code>Graphics.FromImage()</code> y guarda los bytes de la imagen para un informe, una exportación o una descarga.
<code>Invalidate()</code>	La llamada que indica a un control dibujado que su representación ha quedado obsoleta; actualiza el widget del cliente y la superficie del diseñador de Visual Studio, así que corresponde al punto donde cambió el estado, no al interior de los ayudantes de dibujo.

3. Desarrollo de la lección

Cuatro canalizaciones que parecen C# corriente

La primera canalización es el evento `Paint` del control, que admiten todos los controles de Wisej.NET. El controlador recibe un `Wisej.Web.PaintEventArgs` cuya propiedad `Graphics` es una superficie `System.Drawing.Graphics` y cuyo `ClipRectangle` identifica el rectángulo que se está dibujando. Wisej.NET renderiza el fondo dibujado y lo envía al navegador; la documentación de dibujo personalizado describe el resultado como un PNG que se transfiere y después se elimina de la memoria. Eso es lo que hace que una rutina antigua al estilo GDI+ resulte insólitamente reutilizable en una aplicación web.

La segunda canalización es `DataGridView.CellPaint`, especializada en celdas. Una celda o una columna entera debe tener `UserPaint` en `true`; la cuadrícula dispara entonces `CellPaint` con `DataGridViewCellPaintEventArgs`, que lleva los conocidos `Graphics` y `ClipRectangle` más `RowIndex` y `ColumnIndex`. Encaja con elementos visuales compactos —indicadores de estado, gráficos de barras diminutos, sparklines, insignias— y con rutinas que resultarían incómodas de reproducir como marcado.

La tercera canalización es `Wisej.Web.Canvas`, un canvas de navegador de modo inmediato cuya API se parece deliberadamente al modelo `CanvasRenderingContext2D` de HTML5. La cuarta no tiene ningún control de por medio: crea un `Bitmap`, obtén un `Graphics` con `Graphics.FromImage()`, dibuja y guarda o envía la imagen. Wisej.NET 4 incluye `System.Drawing.Managed`, que implementa el conocido espacio de nombres `System.Drawing` en código gestionado sobre los destinos de .NET moderno. En el laboratorio de este módulo las cuatro representan la única `TelemetrySample` que contiene la página.

Dónde se ejecuta el código y qué cruza la red

Un control dibujado en el servidor no envía tu llamada `DrawLine()` al navegador. El servidor ejecuta el dibujo sobre una superficie gráfica y Wisej.NET transfiere la imagen terminada: cómodo, porque el navegador nunca tiene que entender tu algoritmo, y caro cuando la superficie es grande y cambia a menudo, ya que en cada redibujado se pagan CPU de servidor y bytes. Mide eso antes de dejar que un control dibujado se redibuje con un temporizador.

Canvas invierte esto. La API `Canvas` de Wisej representa operaciones del canvas 2D del navegador, y esas operaciones se ejecutan en el cliente. `LiveUpdate` es `false` de forma predeterminada, así que los comandos emitidos durante una petición viajan con la actualización normal de Wisej.NET; ponerlo en `true` los envía de inmediato, lo que encaja con demostraciones incrementales pero no sustituye a la animación en el cliente: un viaje de ida y vuelta al servidor por fotograma es la arquitectura equivocada. El renderizado fuera de pantalla no cruza nada hasta que entregas los bytes, y `CellPaint` usa el mismo mecanismo de imagen que `Paint`, limitado a una celda.

Modelo retenido, píxeles inmediatos

El «modo inmediato» es la parte que suele pillar a la gente. Un Canvas recuerda píxeles, no tus objetos de negocio, y Wisej.NET no reproduce cada línea y cada rectángulo tras una actualización. Mantén el modelo —formas, puntos, el elemento seleccionado, el zoom, el desplazamiento, los colores— en objetos C# corrientes y haz que `Redraw` sea determinista: dado ese modelo y el tamaño actual del canvas, reconstruye el elemento visual completo. En el laboratorio redimensionas el navegador y anotas qué superficies volvieron por sí solas.

Los controladores de Paint merecen la misma disciplina. Trátalos como funciones de renderizado: observan el estado, lo dibujan y no cambian nada. Si la lectura cambia, actualiza el modelo e invalida el control; nunca ajustes la lectura porque se haya solicitado un dibujado. El código de renderizado que muta el estado es la razón por la que un elemento visual acaba dependiendo de cuántas veces se ha redibujado.

Invalidate() en la frontera de un cambio de estado

`Control.Invalidate()` hace que el control actualice su widget en el cliente, y también actualiza la superficie completa en el diseñador de Visual Studio. En un control dibujado por el usuario es la llamada natural cuando cambia una propiedad que afecta al elemento visual, y su lugar natural es el setter de la propiedad, protegido para que un valor sin cambios no cueste nada:

```
private double _reading;

public double Reading
{
    get => _reading;
    set
    {
        value = Math.Clamp(value, 0, 100);
        if (_reading == value)
            return;

        _reading = value;
        this.Invalidate();
    }
}
```

El código de dibujo puede dar ya por supuesto que el valor es válido y limitarse a representar el estado actual. Mantén la invalidación en esas fronteras en lugar de repartirla por ayudantes de bajo nivel, donde se convierte en tormentas de redibujado que nadie puede rastrear. Una restricción de diseño con la que hay que contar: la guía de dibujo personalizado es explícita en que `BackgroundImage` y el dibujo personalizado no se pueden combinar en el mismo control, porque el dibujo toma el control del fondo. Para poner una imagen detrás de tu dibujo, dibújala tú mismo con `Graphics.DrawImage()`.

La escala que deja el dibujo personalizado en último lugar

Los gráficos personalizados deben ser la respuesta a un requisito, nunca un reflejo heredado del trabajo en escritorio. Recorre la escala en orden:

- ¿Puede hacerlo ya un control estándar, un tema, un icono, una imagen, una barra de progreso, una extensión de gráficos o el CSS? Prefiere eso.
- ¿El contenido es HTML por naturaleza y se puede leer como contenido del DOM? Considera el marcado antes que los píxeles.
- ¿Es un elemento visual pequeño que pertenece al fondo de un control y que cambia a velocidad humana? `Paint`.
- ¿Es un elemento visual compacto dentro de una celda de cuadrícula? `CellPaint`, después de compararlo con el HTML de celda.
- ¿Es una escena interactiva o nativa del navegador? `Canvas`. ¿La salida es un archivo en lugar de una pantalla? `System.Drawing.Managed` fuera de pantalla. ¿Es sensible a la velocidad de fotogramas? Deja el bucle en el cliente.

Dos comprobaciones cierran la decisión. Los píxeles no transportan significado para las tecnologías de asistencia, así que todo lo importante que exista solo como dibujo necesita un canal textual paralelo: una etiqueta, un tooltip, `AccessibleName` o `AccessibleDescription`. Y la transferencia por red más la CPU del servidor forman parte del presupuesto de dibujo, igual que las reservas de memoria forman parte del presupuesto de un bucle caliente. Un indicador dibujado que gana en estética y pierde en ambas cosas es una decisión que deberías poder defender en una revisión.



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:

- ✓ Distinguir el Paint de un control, DataGridView.CellPaint, Wisej.Web.Canvas y el renderizado fuera de pantalla con System.Drawing.Managed.
- ✓ Explicar la frontera servidor/navegador de cada técnica de dibujo.
- ✓ Entender cuándo un redibujado envía una imagen y cuándo Canvas envía comandos de dibujo al navegador.
- ✓ Usar Invalidate() de forma deliberada cuando cambia el estado visual.
- ✓ Identificar los casos en los que HTML, CSS, un control estándar de Wisej.NET o una extensión son mejor solución que el dibujo personalizado.

Guía del laboratorio / examen

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



Laboratorio práctico

Construye el shell de VisualOperationsStudio y representa una única lectura de telemetría en cuatro superficies a la vez. Crea una VisualOperationsPage que contenga una sola instancia del modelo TelemetrySample y después presenta su valor como (1) un Label simple más un ProgressBar, (2) un Panel con un controlador de Paint que dibuja una barra a partir de PaintEventArgs.Graphics y e.ClipRectangle, (3) un Wisej.Web.Canvas cuyo controlador de Redraw reconstruye la misma barra a partir del modelo y (4) un Bitmap fuera de pantalla renderizado con Graphics.FromImage y mostrado en un PictureBox. Conecta un botón que cambie el valor del modelo y actualice las cuatro superficies mediante Invalidate() y un redibujado del Canvas; después redimensiona el navegador y anota qué superficies sobreviven y cuáles hay que reconstruir.

Punto de partida

Crea un proyecto nuevo de Wisej.NET 4.x llamado VisualOperationsStudio, o abre el shell vacío si tu instructor te ha proporcionado uno: una página inicial, un tema predeterminado y nada dibujado a mano en ninguna parte. Todo lo de este laboratorio lo añades tú, y las cuatro superficies que construyes aquí son las que todos los módulos posteriores amplían hasta convertirlas en un indicador real, una cuadrícula dibujada, un editor de topología y una exportación a PNG.

Enfoque sugerido

- ✓ Añade una `VisualOperationsPage` y una pequeña clase `TelemetrySample` con el nombre de la máquina, una lectura entre 0 y 100 y una marca de tiempo. Crea exactamente una instancia como campo de la página; todas las superficies deben leer esa instancia y ninguna puede guardar su propia copia.
- ✓ Dispón cuatro áreas etiquetadas en la página. Rellena la primera con un `Label` y un `ProgressBar` enlazados a la lectura, para tener una referencia sin píxeles con la que comparar las otras tres.
- ✓ Coloca un `Panel` en la segunda área y gestiona su evento `Paint`. Dibuja la barra con `e.Graphics`, dimensionala dentro de `e.ClipRectangle` y libera los pens y brushes que crees. Lee la lectura del modelo y no cambies nada en el controlador.
- ✓ Pon un `Wisej.Web.Canvas` en la tercera área y lleva todo su dibujo al controlador de `Redraw`, para que la escena se reconstruya a partir del modelo en lugar de acumularse. En la cuarta área dibuja un `Bitmap` fuera de pantalla con `Graphics.FromImage` y muéstralo en un `PictureBox`.
- ✓ Añade un botón que asigne un valor nuevo al modelo y después actualice todas las superficies: actualiza la etiqueta y la barra de progreso, llama a `Invalidate()` en el panel dibujado, redibuja el canvas y reconstruye la imagen fuera de pantalla dentro de `try / catch`. Después redimensiona la ventana del navegador y escribe tus notas.

Entregables requeridos

- ✓ VisualOperationsPage con un único modelo TelemetrySample del que leen todas las superficies
- ✓ Controlador de Paint que dibuja desde PaintEventArgs.Graphics dentro de e.ClipRectangle
- ✓ Controlador de Redraw del Canvas que reconstruye toda la escena a partir del modelo después de un redimensionado
- ✓ Bitmap fuera de pantalla renderizado con Graphics.FromImage y mostrado en la página
- ✓ SurfaceNotes.md que registre qué cruza la red en cada superficie y qué elementos visuales sobreviven a un redimensionado

Preguntas de repaso

- ✓ Estableces la lectura del modelo desde el botón, pero olvidas la llamada a `Invalidate()` en el panel dibujado. La etiqueta y el canvas se actualizan. Explica exactamente por qué el panel no lo hace y qué está mirando el usuario hasta que algo más fuerce el redibujado.
- ✓ Tu primera versión del Canvas dibuja la barra una sola vez al cargar la página, en lugar de hacerlo desde `Redraw`. Describe qué ve el usuario después de redimensionar el navegador y por qué el `Bitmap` fuera de pantalla de la cuarta área se comporta de otra forma.
- ✓ Quien revisa tu trabajo pregunta por qué está en el laboratorio la primera área, el `Label` y el `ProgressBar` simples. Responde con los costes que pagan las otras tres superficies y ella no, y nombra un requisito que justificaría pagarlos.



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.