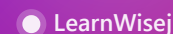


L'architecture de dessin de Wisej.NET

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Quatre pipelines de rendu et comment choisir entre eux : le Paint des contrôles qui expédie une image rendue par le serveur, le CellPaint du DataGridView à l'intérieur d'une seule cellule, Wisej.Web.Canvas qui envoie des commandes de dessin au navigateur, et System.Drawing.Managed hors écran qui produit des octets d'image — plus l'échelle de décision qui garde les contrôles standard et le CSS comme choix par défaut. Ce que cela signifie pour un développeur .NET expérimenté qui construit déjà des écrans Wisej.NET et a désormais besoin de graphiques personnalisés — et comment aborder ce module.



Objectif en production

Quatre pipelines de rendu et comment choisir entre eux : le Paint des contrôles qui expédie une image rendue par le serveur, le CellPaint du DataGridView à l'intérieur d'une seule cellule, Wisej.Web.Canvas qui envoie des commandes de dessin au navigateur, et System.Drawing.Managed hors écran qui produit des octets d'image — plus l'échelle de décision qui garde les contrôles standard et le CSS comme choix par défaut.

1. Le problème en production

VisualOperationsStudio doit afficher la même mesure de machine à quatre endroits : un cadran sur la page d'exploitation, une barre de santé dans une ligne de grille, un nœud sur une carte de topologie, et un PNG joint au rapport de poste du soir. Chacun se résume à « dessiner un nombre », et pourtant, se saisir de la première API de dessin dont on se souvient sur le bureau produit quatre morceaux de code aux comportements radicalement différents en CPU, en bande passante et au redimensionnement. Ce module place le choix de la surface avant le code de dessin, parce que c'est la surface qui décide où les pixels sont produits et de ce qui voyage jusqu'au navigateur.

2. Concepts clés

Concept	Pourquoi c'est important
<code>Paint</code> / <code>PaintEventArgs</code>	L'événement de dessin au niveau du contrôle : votre C# s'exécute sur le serveur sur le <code>Graphics</code> qu'il fournit, à l'intérieur du <code>ClipRectangle</code> qu'il indique, et Wisej.NET transforme le résultat en image pour le navigateur.
<code>DataGridView.CellPaint</code>	Le même mécanisme de dessin, limité à une cellule ; son <code>DataGridViewCellPaintEventArgs</code> dérive de <code>PaintEventArgs</code> et ajoute <code>RowIndex</code> et <code>ColumnIndex</code> .
<code>UserPaint</code>	L'interrupteur qu'une cellule ou une colonne entière doit activer avant que la grille ne déclenche <code>CellPaint</code> pour elle, ce qui rend le dessin explicite colonne par colonne au lieu de l'imposer à toute la grille.
<code>Wisej.Web.Canvas</code>	Une surface 2D du navigateur en mode immédiat, qui reflète le modèle <code>CanvasRenderingContext2D</code> de HTML5 ; les appels de dessin sont exécutés par le client, et non rendus en une image sur le serveur.
<code>Redraw</code>	L'événement du Canvas dans lequel toute la scène est reconstruite depuis votre modèle, parce que le canvas conserve des pixels et les perd dès qu'il est recréé ou redimensionné.
<code>System.Drawing.Managed</code>	Le rendu hors écran, sans contrôle visible : créez un <code>Bitmap</code> , obtenez son <code>Graphics</code> par <code>Graphics.FromImage()</code> , et gardez les octets de l'image pour un rapport, un export ou un téléchargement.
<code>Invalidate()</code>	L'appel qui indique à un contrôle dessiné que son visuel est périmé ; il met à jour le widget client ainsi que la surface du concepteur Visual Studio, sa place est donc là où l'état change, et non dans des helpers de dessin.

3. Le fil de la leçon

Quatre pipelines qui ressemblent tous à du C# ordinaire

Le premier pipeline est l'événement `Paint` du contrôle, que tout contrôle Wisej.NET prend en charge. Le gestionnaire reçoit un `Wisej.Web.PaintEventArgs` dont la propriété `Graphics` est une surface `System.Drawing.Graphics` et dont le `ClipRectangle` identifie le rectangle en cours de dessin. Wisej.NET rend le fond dessiné et l'envoie au navigateur ; la documentation sur le dessin personnalisé décrit le résultat comme un PNG transféré puis retiré de la mémoire. C'est ce qui rend une vieille routine de style GDI+ étonnamment réutilisable dans une application web.

Le deuxième pipeline est `DataGridView.CellPaint`, spécialisé pour les cellules. Une cellule ou une colonne entière doit avoir `UserPaint` à `true` ; la grille déclenche alors `CellPaint` avec un `DataGridViewCellPaintEventArgs`, qui porte les habituels `Graphics` et `ClipRectangle` plus `RowIndex` et `ColumnIndex`. Il convient aux visuels compacts — jauges d'état, minuscules diagrammes en barres, sparklines, pastilles — et aux routines qu'il serait malcommode de reproduire en markup.

Le troisième pipeline est `Wisej.Web.Canvas`, un canvas de navigateur en mode immédiat dont l'API ressemble délibérément au modèle `CanvasRenderingContext2D` de HTML5. Le quatrième ne comporte aucun contrôle : créez un `Bitmap`, obtenez un `Graphics` par `Graphics.FromImage()`, dessinez, puis enregistrez ou envoyez l'image. Wisej.NET 4 livre `System.Drawing.Managed`, qui implémente en code managé le familier espace de noms `System.Drawing` sur les cibles .NET modernes. Dans le lab de ce module, les quatre rendent le seul `TelemetrySample` que détient la page.

Où le code s'exécute et ce qui traverse le réseau

Un contrôle dessiné par le serveur n'envoie pas votre appel `DrawLine()` au navigateur. Le serveur exécute le dessin sur une surface graphique et Wisej.NET transfère l'image finie — commode, parce que le navigateur n'a jamais à comprendre votre algorithme, et coûteux lorsque la surface est grande et change souvent, puisque le CPU serveur et les octets se paient à chaque redessin. Mesurez cela avant de laisser un contrôle dessiné se redessiner sur une minuterie.

Le Canvas inverse la situation. L'API `Canvas` de Wisej représente les opérations du canvas 2D du navigateur, et ces opérations sont exécutées chez le client. `LiveUpdate` vaut `false` par défaut : les commandes émises pendant une requête voyagent donc avec la mise à jour Wisej.NET normale ; le passer à `true` les pousse immédiatement, ce qui convient aux démonstrations progressives mais ne remplace pas une animation côté client — un aller-retour serveur par image est une mauvaise architecture. Le rendu hors écran ne traverse rien tant que vous ne remettez pas les octets, et `CellPaint` utilise le même mécanisme d'image que `Paint`, borné à une cellule.

Modèle conservé, pixels immédiats

Le « mode immédiat » est ce qui prend les gens au dépourvu. Un Canvas mémorise des pixels, pas vos objets métier, et Wisej.NET ne rejoue pas chaque ligne et chaque rectangle après un rafraîchissement. Gardez le modèle — formes, points, élément sélectionné, zoom, panoramique, couleurs — dans des objets C# ordinaires, et rendez `Redraw` déterministe : à partir de ce modèle et de la taille courante du canvas, il reconstruit le visuel complet. Dans le lab, vous redimensionnez le navigateur et notez quelles surfaces sont revenues d'elles-mêmes.

Les gestionnaires Paint méritent la même discipline. Traitez-les comme des fonctions de rendu : observer l'état, le dessiner, ne rien changer. Si la mesure change, mettez le modèle à jour et invalidez le contrôle ; n'ajustez jamais la mesure parce qu'un dessin a été demandé. Du code de rendu qui modifie l'état, c'est ainsi qu'un visuel finit par dépendre du nombre de fois où il a été redessiné.

Invalidate() à la frontière d'un changement d'état

`Control.Invalidate()` amène le contrôle à mettre à jour son widget client, et met également à jour toute la surface dans le concepteur Visual Studio. Dans un contrôle dessiné par votre code, c'est l'appel naturel lorsqu'une propriété qui influence le visuel change, et sa place naturelle est le setter de la propriété, protégé pour qu'une valeur inchangée ne coûte rien :

```
private double _reading;

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

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

Le code de dessin peut désormais supposer que la valeur est valide et se contenter de rendre l'état courant. Gardez l'invalidation à ces frontières plutôt que de la parsemer dans des helpers de bas niveau, où elle se transforme en tempêtes de redessins que personne ne sait tracer. Une contrainte de conception à prévoir : le guide du dessin personnalisé est explicite, `BackgroundImage` et le dessin personnalisé ne peuvent pas être combinés sur un même contrôle, parce que le dessin prend le pas sur l'arrière-plan. Pour une image derrière votre dessin, tracez-la vous-même avec `Graphics.DrawImage()`.

L'échelle qui garde le dessin personnalisé en dernier

Les graphiques personnalisés doivent répondre à un besoin, jamais être un réflexe hérité du travail sur le bureau. Parcourez l'échelle dans l'ordre :

- Un contrôle standard, un thème, une icône, une image, une barre de progression, une extension de graphique ou du CSS savent-ils déjà le faire ? Préférez-les.
- Le contenu est-il naturellement du HTML, lisible en tant que contenu du DOM ? Envisagez le markup avant les pixels.
- Est-ce un petit visuel appartenant à l'arrière-plan d'un contrôle et qui change à vitesse humaine ? `Paint`.
- Est-ce un visuel compact dans une cellule de grille ? `CellPaint`, après l'avoir comparé au HTML de cellule.
- Est-ce une scène interactive ou native au navigateur ? `Canvas`. La sortie est-elle un fichier plutôt qu'un écran ? `System.Drawing.Managed` hors écran. Est-ce sensible à la fréquence d'images ? Gardez la boucle sur le client.

Deux vérifications closent la décision. Les pixels ne portent aucun sens pour les technologies d'assistance : tout ce qui compte et n'existe que sous forme de dessin a donc besoin d'un canal textuel parallèle — un libellé, une info-bulle, `AccessibleName` ou `AccessibleDescription`. Et le transfert réseau ainsi que le CPU serveur font partie du budget de dessin, exactement comme les allocations font partie du budget d'une boucle chaude. Une jauge dessinée qui gagne sur l'apparence et perd sur les deux est une décision que vous devez pouvoir défendre en revue.



Réflexe dessin : la surface d'abord

Choisissez la surface de rendu avant d'écrire le moindre appel de dessin. Demandez-vous où les pixels sont produits et ce qui transite sur le réseau : un gestionnaire Paint de contrôle s'exécute sur le serveur et expédie une image, CellPaint fait de même à l'intérieur d'une cellule de grille, Canvas envoie des commandes de dessin que le navigateur exécute et ne conserve rien après un redimensionnement, et System.Drawing.Managed produit des octets d'image sans aucun contrôle. Gardez le modèle sémantique dans des objets C#, faites de chaque moteur de rendu une fonction déterministe de ce modèle, n'invalidiez que lorsqu'une dépendance visuelle a changé, et laissez toujours un équivalent textuel de tout ce qui compte et n'existe que sous forme de pixels.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Distinguer le Paint des contrôles, DataGridView.CellPaint, Wisej.Web.Canvas et le rendu hors écran avec System.Drawing.Managed.
- ✓ Expliquer la frontière serveur/navigateur de chaque technique de dessin.
- ✓ Comprendre quand un redessin envoie une image et quand le Canvas envoie des commandes de dessin au navigateur.
- ✓ Utiliser Invalidate() délibérément lorsque l'état visuel change.
- ✓ Identifier les cas où le HTML, le CSS, un contrôle Wisej.NET standard ou une extension constituent une meilleure solution que le dessin personnalisé.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Construisez le shell de VisualOperationsStudio et affichez une même mesure de télémétrie sur quatre surfaces à la fois. Créez une VisualOperationsPage qui détient une seule instance du modèle TelemetrySample, puis présentez sa valeur sous la forme (1) d'un simple Label accompagné d'une ProgressBar, (2) d'un Panel doté d'un gestionnaire Paint qui dessine une barre à partir de PaintEventArgs.Graphics et de e.ClipRectangle, (3) d'un Wisej.Web.Canvas dont le gestionnaire Redraw reconstruit la même barre depuis le modèle, et (4) d'un Bitmap hors écran rendu avec Graphics.FromImage et affiché dans une PictureBox. Câblez un bouton qui change la valeur du modèle et rafraîchit les quatre surfaces via Invalidate() et un redessin du Canvas, puis redimensionnez le navigateur et notez quelles surfaces survivent et lesquelles doivent être reconstruites.

Point de départ

Créez un nouveau projet Wisej.NET 4.x nommé VisualOperationsStudio, ou ouvrez le shell vide si votre formateur vous en a fourni un : une page de démarrage, un thème par défaut, et rien de dessiné à la main nulle part. Tout ce qui figure dans ce lab est ajouté par vous, et les quatre surfaces que vous construisez ici sont celles que chaque module suivant transformera en véritable jauge, en grille dessinée, en éditeur de topologie et en export PNG.

Approche suggérée

- ✓ Ajoutez une `VisualOperationsPage` et une petite classe `TelemetrySample` contenant le nom de la machine, une mesure comprise entre 0 et 100 et un horodatage. Créez exactement une instance dans un champ de la page ; chaque surface doit lire cette instance et aucune ne peut en garder sa propre copie.
- ✓ Disposez quatre zones libellées sur la page. Remplissez la première avec un `Label` et une `ProgressBar` liés à la mesure, afin de disposer d'une référence sans pixels à laquelle comparer les trois autres.
- ✓ Posez un `Panel` dans la deuxième zone et gérez son événement `Paint`. Dessinez la barre avec `e.Graphics`, dimensionnez-la à l'intérieur de `e.ClipRectangle`, et libérez les pens et brushes que vous créez. Lisez la mesure depuis le modèle et ne changez rien dans le gestionnaire.
- ✓ Placez un `Wisej.Web.Canvas` dans la troisième zone et déplacez tout son dessin dans le gestionnaire `Redraw`, pour que la scène soit reconstruite depuis le modèle au lieu de s'accumuler. Dans la quatrième zone, rendez un `Bitmap` hors écran avec `Graphics.FromImage` et affichez-le dans une `PictureBox`.
- ✓ Ajoutez un bouton qui affecte une nouvelle valeur au modèle puis rafraîchit chaque surface : mettez à jour le libellé et la barre de progression, appelez `Invalidate()` sur le panneau dessiné, redessinez le canvas, et reconstruisez l'image hors écran dans un `try / catch`. Redimensionnez ensuite la fenêtre du navigateur et rédigez vos notes.

Livrables attendus

- ✓ VisualOperationsPage détenant un unique modèle TelemetrySample que toutes les surfaces lisent
- ✓ Gestionnaire Paint dessinant depuis PaintEventArgs.Graphics à l'intérieur de e.ClipRectangle
- ✓ Gestionnaire Redraw du Canvas qui reconstruit toute la scène depuis le modèle après un redimensionnement
- ✓ Bitmap hors écran rendu avec Graphics.FromImage et affiché dans la page
- ✓ SurfaceNotes.md consignant ce qui transite sur le réseau pour chaque surface et quels visuels survivent à un redimensionnement

Questions de révision

- ✓ Vous définissez la mesure du modèle depuis le bouton mais oubliez l'appel à `Invalidate()` sur le panneau dessiné. Le libellé et le canvas se mettent à jour. Expliquez précisément pourquoi le panneau, lui, ne le fait pas, et ce que l'utilisateur regarde jusqu'à ce qu'autre chose force son redessin.
- ✓ Votre première version du Canvas dessine la barre une seule fois au chargement de la page au lieu de le faire depuis `Redraw`. Décrivez ce que voit l'utilisateur après avoir redimensionné le navigateur, et pourquoi le `Bitmap` hors écran de la quatrième zone se comporte différemment.
- ✓ Votre relecteur demande pourquoi la première zone — le simple `Label` et la `ProgressBar` — figure dans le lab. Répondez par les coûts que les trois autres surfaces paient et qu'elle ne paie pas, et nommez un besoin qui justifierait de les payer.



Réflexe dessin : la surface d'abord

Choisissez la surface de rendu avant d'écrire le moindre appel de dessin. Demandez-vous où les pixels sont produits et ce qui transite sur le réseau : un gestionnaire Paint de contrôle s'exécute sur le serveur et expédie une image, CellPaint fait de même à l'intérieur d'une cellule de grille, Canvas envoie des commandes de dessin que le navigateur exécute et ne conserve rien après un redimensionnement, et System.Drawing.Managed produit des octets d'image sans aucun contrôle. Gardez le modèle sémantique dans des objets C#, faites de chaque moteur de rendu une fonction déterministe de ce modèle, n'invalidiez que lorsqu'une dépendance visuelle a changé, et laissez toujours un équivalent textuel de tout ce qui compte et n'existe que sous forme de pixels.