

Wisej.Web.Canvas et le modèle Canvas 2D de HTML5

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



Guide de la leçon

Lire la documentation Canvas 2D du navigateur et écrire l'équivalent Wisej.NET : des membres en PascalCase sur Wisej.Web.Canvas, des angles en degrés là où le navigateur utilise des radians, Save et Restore autour des transformations, les dégradés, les motifs, les ombres et le découpage, et Redraw comme unique endroit où une scène est reconstruite. 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

Lire la documentation Canvas 2D du navigateur et écrire l'équivalent Wisej.NET : des membres en PascalCase sur Wisej.Web.Canvas, des angles en degrés là où le navigateur utilise des radians, Save et Restore autour des transformations, les dégradés, les motifs, les ombres et le découpage, et Redraw comme unique endroit où une scène est reconstruite.

1. Le problème en production

Tout ce que VisualOperationsStudio a dessiné jusqu'ici a été produit sur le serveur : la jauge du Module 2 et les cellules Health et Trend du Module 3 sont rendues avec `System.Drawing` et parviennent à l'utilisateur sous forme d'images. La vue de topologie dont l'application a désormais besoin, ce sont des dizaines de petits nœuds, d'arêtes et de libellés qu'un utilisateur finira par déplacer, zoomer et sélectionner — exactement le travail qu'un canvas de navigateur fait bien.

`Wisej.Web.Canvas` vous remet cette surface depuis C#, et vous la facture d'un mode de défaillance différent. Les pixels appartiennent au navigateur, rien sur le serveur ne s'en souvient, et un redimensionnement peut les effacer. Du code qui dessine une seule fois paraît parfait dans le concepteur et perd sa scène dès qu'un utilisateur tire le bord d'une fenêtre.

2. Concepts clés

Concept	Pourquoi c'est important
<code>Wisej.Web.Canvas</code>	Un contrôle Wisej qui expose à C# la surface 2D en mode immédiat du navigateur ; ses appels sont des commandes de dessin que le navigateur exécute, et non des opérations <code>System.Drawing.Graphics</code> .
événement <code>Redraw</code>	Le point d'accroche que la surface déclenche lorsqu'elle a de nouveau besoin de son contenu ; comme Wisej.NET ne conserve pas les commandes, le gestionnaire doit reconstruire toute la scène.
correspondance PascalCase	Les membres se transposent par la casse — <code>fillRect()</code> vers <code>FillRect()</code> , <code>strokeStyle</code> vers <code>StrokeStyle</code> — tandis que les valeurs CSS deviennent des types .NET comme <code>Color</code> , <code>Rectangle</code> , <code>Font</code> et <code>Image</code> .
angles en degrés	Wisej.NET documente les arguments d'angle en degrés par simplicité, là où le navigateur utilise des radians, ce qui change chaque appel <code>Arc()</code> et <code>Rotate()</code> porté.
<code>Save()</code> / <code>Restore()</code>	Empilent et dépilent tout l'état de rendu — styles, transformation, découpage, alpha, réglages de ligne et de texte — pour qu'un changement local ne puisse pas fuiter vers la forme suivante.
<code>TextFont</code> , <code>TextAlign</code> , <code>TextBaseline</code>	Le pendant Wisej.NET des <code>font</code> , <code>textAlign</code> et <code>textBaseline</code> du navigateur ; la police est une véritable <code>System.Drawing.Font</code> .
<code>LiveUpdate</code>	À false par défaut : les commandes voyagent alors avec la mise à jour d'interface normale de la requête ; à true, chaque appel est poussé immédiatement, ce qui convient à un dévoilement progressif et ruine une boucle d'animation.

3. Le fil de la leçon

Lisez la documentation du navigateur, puis écrivez le C#

Le Canvas de Wisej.NET est une façade programmable côté serveur au-dessus du contexte 2D du navigateur : on l'apprend donc en traduisant des exemples. L'essentiel est mécanique : `beginPath()` devient `BeginPath()` , `globalAlpha` devient `GlobalAlpha` , `setLineDash()` devient `SetLineDash()` . Là où le navigateur attend une chaîne CSS, Wisej.NET attend une valeur .NET : une couleur est un `Color` , une police une `Font` , et une image un `Image` ou une chaîne de source d'image.

Ce qu'une référence de navigateur ne peut pas vous dire, c'est que la liste des membres documentés de Wisej.NET ne couvre pas toute l'API Canvas moderne. `measureText` , `getImageData` , `roundRect` , `ellipse` et `createConicGradient` n'y figurent pas. Traduire comporte donc une étape supplémentaire : confirmer que le membre existe dans la version sur laquelle vous compilez. Quand ce n'est pas le cas, la réponse honnête est un Widget ou du JavaScript client, jamais un nom de méthode deviné.

Redraw est le seul endroit où votre scène existe

Les appels de dessin du Canvas ne sont pas conservés par Wisej.NET. Les commandes sont émises, le navigateur les peint, et l'implication du serveur s'arrête là. Lorsque la surface est redimensionnée ou rafraîchie, le bitmap peut être effacé, et aucune copie en cache n'existe pour le rétablir. Ce seul fait sépare le code Canvas qui fonctionne de celui qui semble seulement fonctionner.

La scène vit donc dans des objets C#, une méthode la reconstruit intégralement, et `Redraw` appelle cette méthode. Dans le lab, `DrawPlayground` commence par un `ClearRect` sur toute la surface et réémet chaque commande, ce qui lui donne le même résultat au premier affichage, après un changement de valeur et après un redimensionnement.

```
private void canvasPlayground_Redraw(object sender, EventArgs e) => DrawPlayground();

private void DrawPlayground()
{
    canvasPlayground.ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height);

    canvasPlayground.Save();
    canvasPlayground.Translate(240, 160);
    canvasPlayground.Rotate(45); // browser: ctx.rotate(Math.PI / 4)
    canvasPlayground.FillStyle = Color.Coral;
    canvasPlayground.FillRect(-60, -24, 120, 48);
    canvasPlayground.Restore();
}
```

Deux habitudes cassent tout cela. Dessiner dans `Load` ne s'exécute qu'une fois : la scène disparaît après le premier redimensionnement. Un gestionnaire qui ne redessine que ce qui a changé est pire : il fonctionne pendant le développement et laisse une surface à moitié vide devant un utilisateur.

Tracés, remplissages, texte, dégradés et ombres

Un tracé se construit d'abord et se peint ensuite. `MoveTo()` et `LineTo()` ajoutent de la géométrie mais ne peignent rien ; `Stroke()` ou `Fill()` se chargent de peindre, avec les `StrokeStyle` , `LineWidth` , `LineCap` et `LineJoin` courants. `BeginPath()` démarre un tracé indépendant, et l'omettre est le bug de portage classique : un contour ultérieur englobe discrètement la forme précédente. `FillRect()` , `StrokeRect()` et `ClearRect()` agissent immédiatement, tandis que `Rect()` s'ajoute au tracé que vous construisez.

`CreateLinearGradient()` et `CreateRadialGradient()` renvoient des objets que vous affectez à `FillStyle` ou `StrokeStyle` , construits à partir d'un tableau d'arrêts de couleur, et `CreatePattern()` prend un `Image` ou une source d'image plus un `CanvasRepeatPattern` . Le texte passe par `TextFont` , `TextAlign` et `TextBaseline` avant `FillText()` ; les images par les surcharges de `DrawImage()` . La profondeur vient des propriétés d'ombre, de `GlobalAlpha` et de `GlobalCompositeOperation` — la composition est subtile et dépend du navigateur, testez donc tout ce que vous livrez avec elle.

Save, Restore et la pile de transformations

L'état de rendu d'un Canvas est bien plus large que la transformation, et `Save()` empile le tout tandis que `Restore()` le dépile. `Translate()` , `Rotate()` et `Scale()` modifient la transformation courante,

`Transform()` y compose une autre matrice, et `SetTransform()` remet à l'identité et applique celle que vous fournissez.

Encadrez chaque nœud : sauvegarder, positionner, dessiner, restaurer. Sans la restauration, toute forme ultérieure hérite du système de coordonnées décalé et la scène dérive d'une manière exaspérante à déboguer. Un découpage se comporte à l'identique, puisqu'il fait partie de l'état courant : un découpage temporaire, c'est donc toujours sauvegarder, tracer, `Clip()`, dessiner, restaurer. Un seul `Restore()` rétablit :

- `FillStyle`, `StrokeStyle` et `GlobalAlpha`
- la matrice de transformation courante et la région de découpe active
- `LineWidth`, `LineCap`, `LineJoin` et le motif de tirets
- les réglages de texte et les propriétés d'ombre

Des degrés plutôt que des radians, et le moment où `LiveUpdate` mérite sa place

La documentation de Wisej.NET sur le dessin personnalisé signale un écart délibéré : les arguments d'angle sont exprimés en degrés. Un exemple qui écrit `ctx.rotate(Math.PI / 4)` devient `Rotate(45)`, et un cercle allant de `0` à `Math.PI * 2` devient `0` à `360` dans `Arc()`. Recopier la valeur en radians compile parfaitement et fait pivoter la forme d'une fraction de degré : c'est pourquoi le lab demande d'écrire la conversion dans un commentaire à côté de la ligne portée.

La dernière différence documentée concerne la livraison. `LiveUpdate` vaut `false` par défaut : les commandes émises pendant une requête voyagent donc avec la mise à jour d'interface de cette requête — un aller-retour, un redessin. À `true`, chaque appel est poussé immédiatement, ce qui est réellement utile quand une longue opération côté serveur doit dévoiler ses étapes, et coûteux partout ailleurs. Ce n'est pas une voie vers l'animation pilotée par le serveur. Laissez-le désactivé pour le rendu principal, afin que le coût soit un choix.



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'invalidez 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 :

- ✓ Expliquer la correspondance entre `CanvasRenderingContext2D` et `Wisej.Web.Canvas`.
- ✓ Utiliser `Redraw` comme point d'accroche canonique de reconstruction de la scène.

- ✓ Dessiner des tracés, des rectangles, du texte, des images, des dégradés, des motifs, des ombres et du contenu composé.
- ✓ Utiliser Save/Restore et les méthodes de transformation en toute sûreté.
- ✓ Comprendre les différences documentées comme le nommage C# et les angles en degrés.

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 dans VisualOperationsStudio une page bac à sable Canvas qui porte une série d'exemples Canvas 2D du navigateur vers l'API Wisej.Web.Canvas documentée. Pilotez le tout depuis une méthode DrawPlayground appelée par l'événement Redraw du Canvas, afin que la scène revienne après un redimensionnement, et commencez-la par un ClearRect sur toute la surface. Dessinez un tracé en contour, des rectangles pleins, une légende placée, un dégradé linéaire et un dégradé radial, puis encadrez chaque section transformée par Save() et Restore() et démontrez Translate, Scale, Rotate et SetTransform. Ajoutez une région découpée, une section GlobalAlpha, une ligne pointillée et une ombre, portez un exemple du navigateur qui pivote de $\text{Math.PI} / 4$ vers la valeur en degrés attendue par Wisej, et laissez LiveUpdate désactivé pour le rendu principal tout en le montrant dans un petit exemple progressif.

Point de départ

Ouvrez VisualOperationsStudio tel que le Module 3 l'a laissé : le contrôle `TelemetryGauge` dessine sur le serveur, `gridOperations` a ses colonnes Health et Trend dessinées par votre code, et la page du Module 1 affiche toujours une valeur de télémétrie sur quatre surfaces. Ajoutez une nouvelle `CanvasPlaygroundPage` contenant un unique `Wisej.Web.Canvas` ancré en remplissage, plus un petit panneau de boutons ; rien sur cette page ne dessine encore quoi que ce soit.

Approche suggérée

- ✓ Écrivez d'abord `DrawPlayground()` et abonnez-la à `canvasPlayground.Redraw` dans le constructeur. Ouvrez la méthode par `ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height)`, dessinez une polyligne en contour avec `BeginPath`, `MoveTo`, `LineTo` et `Stroke`, exécutez la page et redimensionnez le navigateur avant d'ajouter quoi que ce soit d'autre.
- ✓ Ajoutez la section des remplissages : deux formes `FillRect`, une légende placée avec `TextFont`, `TextAlign` et `TextBaseline`, puis une barre remplie par `CreateLinearGradient` et une sphère remplie par `CreateRadialGradient`. Vérifiez chaque membre dans l'API Canvas de Wisej.NET avant de le taper.
- ✓ Construisez la rangée des transformations en quatre blocs encadrés. Chacun est un `Save()`, puis `Translate`, `Scale`, `Rotate` ou `SetTransform`, puis une petite forme libellée, puis `Restore()`. Prenez un exemple du navigateur qui appelle `ctx.rotate(Math.PI / 4)`, écrivez `Rotate(45)`, et placez la conversion dans un commentaire sur la même ligne.
- ✓ Ajoutez la section d'état : une région découpée construite comme un tracé suivi de `Clip()` à l'intérieur d'une paire save/restore, un bloc dessiné sous un `GlobalAlpha` réduit, un filet pointillé utilisant `SetLineDash`, et une carte dessinée avec `ShadowColor`, `ShadowBlur` et les décalages d'ombre. Mettez volontairement un `Restore()` en commentaire, observez le reste de la scène se décaler, puis remettez-le.

- ✓ Laissez `LiveUpdate` à false pour `DrawPlayground` . Donnez à un bouton une courte boucle côté serveur qui active `LiveUpdate` , dessine une étape de progression par itération puis le désactive à nouveau, et notez la différence de trafic ainsi que les cas où ce compromis vaut la peine.

Livrables attendus

- ✓ Bac à sable Canvas dont toute la scène est dessinée par une méthode `DrawPlayground` appelée depuis `Redraw`
- ✓ Tracés, rectangles, légende placée, dégradé linéaire et dégradé radial n'utilisant que des membres documentés de `Wisej.Web.Canvas`
- ✓ `Save()` et `Restore()` autour de chaque section `Translate`, `Scale`, `Rotate` et `SetTransform`
- ✓ Découpage, alpha, ligne pointillée et ombre, plus un portage radians vers degrés écrit noir sur blanc dans un commentaire
- ✓ Une démonstration de `LiveUpdate` tenue à l'écart du rendu principal, avec une note sur les cas où elle est appropriée

Questions de révision

- ✓ Votre bac à sable paraît correct jusqu'à ce qu'un collègue redimensionne le navigateur, après quoi seule la dernière forme dessinée reste visible. Que cela vous apprend-il sur l'origine du reste de la scène ?
- ✓ Vous avez besoin de la largeur en pixels d'une légende pour dimensionner une pastille à sa mesure. L'API Canvas de `Wisej.NET` ne liste aucun membre de mesure de texte — décrivez deux façons d'obtenir malgré tout la mise en page voulue, et ce que chacune coûte.
- ✓ Un relecteur demande pourquoi le découpage de votre section d'état est encadré par `Save()` et `Restore()` alors que les formes qui suivent sont dessinées tout à fait ailleurs. Répondez-lui.



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'invalidez 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.