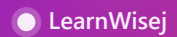


Wisej.Web.Canvas und das HTML5-Canvas-2D-Modell

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Browser-Dokumentation zu Canvas 2D lesen und das Wisej.NET-Äquivalent schreiben: Member in PascalCase auf Wisej.Web.Canvas, Winkel in Grad, wo der Browser Radian verwendet, Save und Restore um Transformationen herum, Farbverläufe, Muster, Schatten und Clipping – und Redraw als die eine Stelle, an der eine Szene neu aufgebaut wird. Was das für erfahrene .NET-Entwickler bedeutet, die bereits Wisej.NET-Oberflächen bauen und nun eigene Grafiken brauchen – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Browser-Dokumentation zu Canvas 2D lesen und das Wisej.NET-Äquivalent schreiben: Member in PascalCase auf Wisej.Web.Canvas, Winkel in Grad, wo der Browser Radian verwendet, Save und Restore um Transformationen herum, Farbverläufe, Muster, Schatten und Clipping – und Redraw als die eine Stelle, an der eine Szene neu aufgebaut wird.

1. Das Problem im Produktivbetrieb

Alles, was VisualOperationsStudio bisher gezeichnet hat, entstand auf dem Server: Das Gauge aus Modul 2 und die Zellen Health und Trend aus Modul 3 werden mit `System.Drawing` gerendert und erreichen die Benutzerin als Bilder. Die Topologieansicht, die die Anwendung als Nächstes braucht, besteht aus Dutzenden kleiner Knoten, Kanten und Beschriftungen, die eine Benutzerin irgendwann verschieben, zoomen und auswählen wird – genau die Arbeit, die eine Browser-Canvas gut erledigt.

`Wisej.Web.Canvas` übergibt Ihnen diese Oberfläche aus C# heraus und berechnet dafür eine andere Fehlerart. Die Pixel gehören dem Browser, nichts auf dem Server merkt sie sich, und eine Größenänderung kann sie auslöschen. Code, der einmal zeichnet, sieht im Designer perfekt aus und verliert seine Szene, sobald eine Benutzerin zum ersten Mal an einer Fensterkante zieht.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
<code>Wisej.Web.Canvas</code>	Ein Wisej-Control, das die 2D-Oberfläche des Browsers im Immediate Mode für C# zugänglich macht; seine Aufrufe sind Zeichenbefehle, die der Browser ausführt, keine Operationen auf <code>System.Drawing.Graphics</code> .
<code>Redraw</code> -Event	Der Haken, den die Oberfläche auslöst, wenn sie ihren Inhalt erneut braucht; weil Wisej.NET die Befehle nicht aufbewahrt, muss der Handler die gesamte Szene neu aufbauen.
Entsprechung in PascalCase	Die Member bilden sich über die Schreibweise ab — <code>fillRect()</code> auf <code>FillRect()</code> , <code>strokeStyle</code> auf <code>StrokeStyle</code> —, während aus CSS-Werten .NET-Typen wie <code>Color</code> , <code>Rectangle</code> , <code>Font</code> und <code>Image</code> werden.
Winkel in Grad	Wisej.NET dokumentiert Winkelargumente der Einfachheit halber in Grad, wo der Browser Radiant verwendet – das ändert jeden portierten Aufruf von <code>Arc()</code> und <code>Rotate()</code> .
<code>Save()</code> / <code>Restore()</code>	Legen den gesamten Rendering-Zustand ab und holen ihn zurück — Stile, Transformation, Clipping, Alpha, Linien- und Texteingstellungen —, damit eine lokale Änderung nicht in die nächste Form überläuft.
<code>TextFont</code> , <code>TextAlign</code> , <code>TextBaseline</code>	Der Wisej.NET-Ersatz für <code>font</code> , <code>textAlign</code> und <code>textBaseline</code> des Browsers; die Schrift ist ein echter <code>System.Drawing.Font</code> .
<code>LiveUpdate</code>	Standardmäßig false, sodass die Befehle mit der normalen UI-Aktualisierung der Anfrage reisen; true sendet jeden Aufruf sofort, was zu einer schrittweisen Enthüllung passt und eine Animationsschleife ruiniert.

3. Der rote Faden der Lektion

Erst die Browser-Dokumentation lesen, dann das C# schreiben

Die Canvas von Wisej.NET ist eine vom Server programmierbare Fassade über dem 2D-Kontext des Browsers – man lernt sie also, indem man Beispiele übersetzt. Das meiste davon ist mechanisch: Aus `beginPath()` wird `BeginPath()`, aus `globalAlpha` wird `GlobalAlpha`, aus `setLineDash()` wird `SetLineDash()`. Wo der Browser eine CSS-Zeichenfolge erwartet, erwartet Wisej.NET einen .NET-Wert: Eine Farbe ist ein `Color`, eine Schrift ein `Font` und ein Bild ein `Image` oder eine Bildquellen-Zeichenfolge.

Die Hälfte, die eine Browser-Referenz Ihnen nicht sagen kann, ist: Die dokumentierte Memberliste von Wisej.NET ist nicht die gesamte moderne Canvas-API. `measureText`, `getImageData`, `roundRect`, `ellipse` und `createConicGradient` tauchen darin nicht auf. Zum Übersetzen gehört deshalb ein zusätzlicher Schritt: Prüfen Sie, ob der Member in der Version existiert, gegen die Sie bauen. Tut er es nicht, lautet die ehrliche Antwort: ein Widget oder Client-JavaScript – nie ein geratener Methodenname.

Redraw ist der einzige Ort, an dem Ihre Szene existiert

Canvas-Zeichenaufrufe werden von Wisej.NET nicht aufbewahrt. Die Befehle werden abgesetzt, der Browser zeichnet sie, und damit endet die Beteiligung des Servers. Wird die Oberfläche in der Größe geändert oder aktualisiert, kann die Bitmap gelöscht werden, und es existiert keine zwischengespeicherte Kopie, um sie zurückzuholen. Diese eine Tatsache trennt funktionierenden Canvas-Code von Code, der nur scheinbar funktioniert.

Die Szene lebt also in C#-Objekten, eine Methode baut sie vollständig neu auf, und **Redraw** ruft diese Methode auf. Im Lab beginnt **DrawPlayground** mit **ClearRect** über die gesamte Fläche und setzt jeden Befehl erneut ab – deshalb liefert es bei der ersten Anzeige, nach einer Wertänderung und nach einer Größenänderung dasselbe Ergebnis.

```
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();
}
```

Zwei Gewohnheiten brechen das. In **Load** zu zeichnen läuft genau einmal, also ist die Szene nach der ersten Größenänderung weg. Ein Handler, der nur das Geänderte neu zeichnet, ist schlimmer: Während der Entwicklung funktioniert er und hinterlässt vor einer Benutzerin eine halb leere Fläche.

Pfade, Füllungen, Text, Farbverläufe und Schatten

Ein Pfad wird zuerst gebaut und danach gezeichnet. **MoveTo()** und **LineTo()** fügen Geometrie hinzu, zeichnen aber nichts; **Stroke()** oder **Fill()** übernimmt das Zeichnen mit dem aktuellen **StrokeStyle**, **LineWidth**, **LineCap** und **LineJoin**. **BeginPath()** beginnt einen eigenständigen Pfad, und es wegzulassen ist der klassische Portierungsfehler: Ein späteres Stroke schließt still die vorherige Form mit ein. **FillRect()**, **StrokeRect()** und **ClearRect()** wirken sofort, während **Rect()** dem Pfad hinzufügt, den Sie gerade aufbauen.

CreateLinearGradient() und **CreateRadialGradient()** geben Objekte zurück, die Sie **FillStyle** oder **StrokeStyle** zuweisen und die aus einem Array von Farbstopps entstehen; **CreatePattern()** nimmt ein **Image** oder eine Bildquelle plus ein **CanvasRepeatPattern** entgegen. Text läuft über **TextFont**, **TextAlign** und **TextBaseline**, bevor **FillText()** zeichnet; Bilder über die **DrawImage()** - Überladungen. Tiefe entsteht über die Schatten-Properties, **GlobalAlpha** und **GlobalCompositeOperation** — Compositing ist subtil und browserabhängig, testen Sie also alles, was Sie damit ausliefern.

Save, Restore und der Transformationsstapel

Der Rendering-Zustand einer Canvas ist weit größer als die Transformation, und `Save()` legt ihn vollständig ab, während `Restore()` ihn zurückholt. `Translate()`, `Rotate()` und `Scale()` verändern die aktuelle Transformation, `Transform()` setzt eine weitere Matrix darauf, und `SetTransform()` setzt auf die Identität zurück und wendet die von Ihnen übergebene an.

Klammern Sie jeden Knoten ein: sichern, positionieren, zeichnen, wiederherstellen. Ohne das Wiederherstellen erbt jede spätere Form das verschobene Koordinatensystem, und die Szene wandert auf eine Weise, die beim Debuggen zur Verzweiflung treibt. Ein Clipping verhält sich genauso, denn es wird Teil des aktuellen Zustands – ein vorübergehendes Clipping heißt also immer: sichern, Pfad, `Clip()`, zeichnen, wiederherstellen. Ein einziges `Restore()` holt zurück:

- `FillStyle`, `StrokeStyle` und `GlobalAlpha`
- die aktuelle Transformationsmatrix und den aktiven Clipping-Bereich
- `LineWidth`, `LineCap`, `LineJoin` und das Strichmuster
- die Text Einstellungen und die Schatten-Properties

Grad statt Radiant – und wann sich LiveUpdate verdient macht

Die Wisej.NET-Dokumentation zum eigenen Zeichnen nennt eine bewusste Abweichung: Winkelargumente werden in Grad ausgedrückt. Aus einem Beispiel mit `ctx.rotate(Math.PI / 4)` wird `Rotate(45)`, und aus einem Kreis von `0` bis `Math.PI * 2` wird in `Arc()` `0` bis `360`. Den Radiantwert zu übernehmen kompiliert einwandfrei und dreht die Form um den Bruchteil eines Grads – deshalb verlangt das Lab die Umrechnung als Kommentar neben der portierten Zeile.

Der letzte dokumentierte Unterschied ist die Auslieferung. `LiveUpdate` ist standardmäßig false, sodass die während einer Anfrage abgesetzten Befehle mit der UI-Aktualisierung dieser Anfrage reisen: ein Roundtrip, ein Neuzeichnen. True sendet jeden Aufruf sofort, was wirklich nützlich ist, wenn eine lange serverseitige Operation ihre Schritte zeigen soll, und überall sonst teuer. Es ist kein Weg zu servergetriebener Animation. Lassen Sie es für das Hauptrendering aus, damit die Kosten eine Entscheidung bleiben, die Sie getroffen haben.



Gewohnheit: erst die Oberfläche, dann das Zeichnen

Wählen Sie die Rendering-Oberfläche, bevor Sie einen einzigen Zeichenaufruf schreiben. Fragen Sie, wo die Pixel entstehen und was über das Netzwerk geht: Der Paint-Handler eines Controls läuft auf dem Server und liefert ein Bild aus, CellPaint tut dasselbe innerhalb einer Grid-Zelle, Canvas schickt Zeichenbefehle, die der Browser ausführt, und behält über eine Größenänderung hinweg nichts, und System.Drawing.Managed erzeugt Bild-Bytes ganz ohne Control. Halten Sie das semantische Modell in C#-Objekten, machen Sie jeden Renderer zu einer deterministischen Funktion dieses Modells, invalidieren Sie nur, wenn sich eine visuelle Abhängigkeit geändert hat, und lassen Sie immer ein textliches Äquivalent für alles Wichtige zurück, das nur als Pixel existiert.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Die Entsprechung zwischen `CanvasRenderingContext2D` und `Wisej.Web.Canvas` erklären.
- ✓ `Redraw` als kanonischen Haken für den Neuaufbau der Szene verwenden.
- ✓ Pfade, Rechtecke, Text, Bilder, Farbverläufe, Muster, Schatten und zusammengesetzte Inhalte zeichnen.
- ✓ `Save/Restore` und die Transformationsmethoden sicher einsetzen.
- ✓ Dokumentierte Unterschiede wie die C#-Benennung und Winkel in Grad verstehen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Bauen Sie in VisualOperationsStudio eine Canvas-Spielwiese, die eine Reihe von Canvas-2D-Beispielen aus dem Browser auf die dokumentierte Wisej.Web.Canvas-API portiert. Steuern Sie alles über eine Methode DrawPlayground, die vom Redraw-Event der Canvas aufgerufen wird, damit die Szene nach einer Größenänderung zurückkehrt, und beginnen Sie sie mit ClearRect über die gesamte Fläche. Rendern Sie einen umrissenen Pfad, gefüllte Rechtecke, eine platzierte Beschriftung, einen linearen und einen radialen Farbverlauf, klammern Sie dann jeden transformierten Abschnitt in Save() und Restore() und führen Sie Translate, Scale, Rotate und SetTransform vor. Ergänzen Sie einen beschnittenen Bereich, einen Alpha-Abschnitt, eine gestrichelte Linie und einen Schatten, portieren Sie ein Browser-Beispiel, das um $\text{Math.PI} / 4$ dreht, in den Gradwert, den Wisej erwartet, und lassen Sie LiveUpdate für das Hauptrendering ausgeschaltet, während Sie es in einem kleinen fortlaufenden Beispiel zeigen.

Ausgangspunkt

Öffnen Sie VisualOperationsStudio in dem Stand, den Modul 3 hinterlassen hat: Das Control `TelemetryGauge` zeichnet auf dem Server, `gridOperations` hat selbst gezeichnete Spalten Health und Trend, und die Seite aus Modul 1 zeigt weiterhin einen Telemetrierwert auf vier Oberflächen. Fügen Sie eine neue `CanvasPlaygroundPage` hinzu, die eine einzelne `Wisej.Web.Canvas` mit Dock-Füllung hält, dazu ein kleines Panel mit Buttons; noch zeichnet auf dieser Seite nichts.

Vorgeschlagenes Vorgehen

- ✓ Schreiben Sie zuerst `DrawPlayground()` und hängen Sie die Methode im Konstruktor an `canvasPlayground.Redraw`. Beginnen Sie die Methode mit `ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height)`, zeichnen Sie einen umrissenen Linienzug mit `BeginPath`, `MoveTo`, `LineTo` und `Stroke`, führen Sie die Seite aus und ändern Sie die Browsergröße, bevor Sie irgendetwas anderes ergänzen.
- ✓ Ergänzen Sie den Füllabschnitt: zwei `FillRect`-Formen, eine mit `TextFont`, `TextAlign` und `TextBaseline` platzierte Beschriftung, danach einen aus `CreateLinearGradient` gefüllten Balken und eine aus `CreateRadialGradient` gefüllte Kugel. Prüfen Sie jeden Member gegen die Canvas-API von Wisej.NET, bevor Sie ihn tippen.
- ✓ Bauen Sie die Transformationsreihe als vier eingeklammerte Blöcke. Jeder besteht aus `Save()`, dann `Translate`, `Scale`, `Rotate` oder `SetTransform`, dann einer kleinen beschrifteten Form und dann `Restore()`. Nehmen Sie ein Browser-Beispiel, das `ctx.rotate(Math.PI / 4)` aufruft, schreiben Sie `Rotate(45)` und setzen Sie die Umrechnung als Kommentar in dieselbe Zeile.
- ✓ Ergänzen Sie den Zustandsabschnitt: einen beschnittenen Bereich, gebaut als Pfad gefolgt von `Clip()` innerhalb eines Save/Restore-Paares, einen Block unter reduziertem `GlobalAlpha`, eine gestrichelte Linie mit `SetLineDash` und eine Karte mit `ShadowColor`, `ShadowBlur` und den

Schattenversätzen. Kommentieren Sie ein `Restore()` absichtlich aus, beobachten Sie, wie sich der Rest der Szene verschiebt, und setzen Sie es dann wieder ein.

- ✓ Lassen Sie `LiveUpdate` für `DrawPlayground` auf false. Geben Sie einem Button eine kurze serverseitige Schleife, die `LiveUpdate` einschaltet, pro Durchlauf einen Fortschrittsschritt zeichnet und es wieder ausschaltet; halten Sie dann den Unterschied im Datenverkehr fest und die Fälle, in denen sich dieser Handel lohnt.

Geforderte Ergebnisse

- ✓ Canvas-Spielwiese, deren gesamte Szene von einer aus `Redraw` aufgerufenen Methode `DrawPlayground` gezeichnet wird
- ✓ Pfade, Rechtecke, eine platzierte Beschriftung, ein linearer und ein radialer Farbverlauf ausschließlich mit dokumentierten `Wisej.Web.Canvas`-Members
- ✓ `Save()` und `Restore()` um jeden Abschnitt mit `Translate`, `Scale`, `Rotate` und `SetTransform`
- ✓ Clipping, Alpha, eine gestrichelte Linie und ein Schatten, dazu eine als Kommentar ausgeschriebene Umrechnung von Radiant in Grad
- ✓ Eine `LiveUpdate`-Demonstration getrennt vom Hauptrendering, mit einer Notiz, wann sie angebracht ist

Wiederholungsfragen

- ✓ Ihre Spielwiese sieht richtig aus, bis eine Kollegin die Browsergröße ändert – danach ist nur noch die zuletzt gezeichnete Form sichtbar. Was sagt Ihnen das darüber, woher der Rest der Szene kam?
- ✓ Sie brauchen die Pixelbreite einer Beschriftung, damit ein Badge passend bemessen werden kann. Die Canvas-API von `Wisej.NET` führt keinen Member zur Textvermessung auf — beschreiben Sie zwei Wege, das gewünschte Layout dennoch zu erreichen, und was jeder davon kostet.
- ✓ Ein Reviewer fragt, warum das Clipping in Ihrem Zustandsabschnitt in `Save()` und `Restore()` eingeklammert ist, obwohl die Formen danach an ganz anderer Stelle gezeichnet werden. Antworten Sie ihm.



Gewohnheit: erst die Oberfläche, dann das Zeichnen

Wählen Sie die Rendering-Oberfläche, bevor Sie einen einzigen Zeichenaufruf schreiben. Fragen Sie, wo die Pixel entstehen und was über das Netzwerk geht: Der `Paint`-Handler eines Controls läuft auf dem Server und liefert ein Bild aus, `CellPaint` tut dasselbe innerhalb einer Grid-Zelle, `Canvas` schickt Zeichenbefehle, die der Browser ausführt, und behält über eine Größenänderung hinweg nichts, und `System.Drawing.Managed` erzeugt Bild-Bytes ganz ohne Control. Halten Sie das semantische Modell in C#-Objekten, machen Sie jeden Renderer zu einer deterministischen Funktion dieses Modells, invalidieren Sie nur, wenn sich eine visuelle Abhängigkeit geändert hat, und lassen Sie immer ein textliches Äquivalent für alles Wichtige zurück, das nur als Pixel existiert.