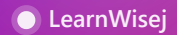


Das Paint-Event von Controls meistern

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Server-seitig gezeichnete Controls, sauber gemacht: PaintEventArgs.Graphics und ClipRectangle, Geometrie getrennt vom Zeichnen, das Freigeben jedes selbst erzeugten Pen, Brush und Font, Property-Setter, die Invalidate() genau einmal aufrufen, und das Kostenbudget fürs Zeichnen, das entscheidet, wann ein gethemtes Control die bessere Antwort ist. 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

Server-seitig gezeichnete Controls, sauber gemacht: PaintEventArgs.Graphics und ClipRectangle, Geometrie getrennt vom Zeichnen, das Freigeben jedes selbst erzeugten Pen, Brush und Font, Property-Setter, die Invalidate() genau einmal aufrufen, und das Kostenbudget fürs Zeichnen, das entscheidet, wann ein gethemtes Control die bessere Antwort ist.

1. Das Problem im Produktivbetrieb

Modul 1 hat einen Telemetriewert auf vier Oberflächen gebracht, damit Sie sehen, wo jedes Pixel entsteht. VisualOperationsStudio will diesen Messwert nun an sechs Stellen, und einen **Paint**-Handler in sechs Panels zu kopieren ist der Weg, auf dem eine Oberfläche mit sechs leicht unterschiedlichen Gauges endet, mit sechs Pens, die nie freigegeben werden, und mit einem Neuzeichnen, sobald sich irgendein unbeteiligtes Feld ändert. Die Lösung ist, keine Handler mehr zu schreiben, sondern ein Control: ein einziges **TelemetryGauge**, dessen öffentliche Properties das Einzige sind, was sein Bild ändern kann, und dessen Kosten Sie in Kilobyte beziffern können.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
<code>Wisej.Web.PaintEventArgs</code>	Das Argument des <code>Paint</code> -Events; es übergibt Ihnen ein <code>System.Drawing.Graphics</code> , das Sie nicht erzeugt haben, und ein Rechteck, in das Sie zeichnen – und Ihr Code läuft auf dem Server.
<code>ClipRectangle</code>	Der Bereich, den Wisej.NET für diesen Zeichenvorgang bereitgestellt hat; das Layout darf die Clientgröße des Controls heranziehen, aber das Zeichnen bleibt an das übergebene Rechteck gebunden.
Helfer <code>GaugeGeometry</code>	Eine reine Funktion aus einem <code>Rectangle</code> plus einem normalisierten Wert zu Ringgrenzen, Winkelbereichen, Zeigerpunkten und einem Beschriftungsrechteck, sodass die Mathematik ohne lebendes Control geprüft werden kann.
<code>Invalidate()</code>	Der eine Aufruf, der Wisej.NET bittet, die Oberfläche des Controls neu zu rendern; ein Property-Setter macht das genau einmal und nur, wenn die Änderung ein Pixel verändern kann.
Ressourcenbesitz	Der Handler gibt jeden <code>Pen</code> , <code>Brush</code> , <code>Font</code> , <code>GraphicsPath</code> und jede <code>Bitmap</code> frei, die er erzeugt, und gibt niemals das vom Framework gelieferte <code>Graphics</code> frei.
Kostenbudget fürs Zeichnen	Drei Zahlen pro gezeichnetem Control: Server-CPU fürs Zeichnen, Serverspeicher für das temporäre Bild und die Bytes, die der Browser laden und dekodieren muss.
<code>AccessibleDescription</code>	Trägt denselben Messwert als Text, sodass ein Wert, der als gezeichneter Bogen existiert, auch für eine Benutzerin verfügbar bleibt, die den Bogen nie sieht.

3. Der rote Faden der Lektion

Der Handler, die Oberfläche und das Rechteck, das Sie bekommen haben

Ein `Paint` -Handler in Wisej.NET wirkt vertraut, weil das Zeichenobjekt tatsächlich ein `System.Drawing.Graphics` ist – aber die Laufzeit darum herum ist kein Desktop. Ihr Handler läuft auf dem Server, und das erzeugte Bild wird zu dem Hintergrund, den der Browser anzeigt. Das nützliche Denkmodell lautet: Sie zeichnen eine serverseitige Bitmap – auch deshalb hat jedes Neuzeichnen einen Netzwerkpreis.

Das Event liefert `Wisej.Web.PaintEventArgs` , und zwei Member tragen alles, was Sie brauchen.

`Graphics` ist die aktive Oberfläche: Sie haben sie nicht belegt und dürfen sie nicht freigeben.

`ClipRectangle` ist der Bereich, den Wisej.NET für diesen Vorgang bereitgestellt hat. Denken Sie beim Layout mit dem Client-Rechteck des Controls, halten Sie das Zeichnen aber an das Rechteck gebunden, das Sie in der Hand haben.

Damit sind die ersten beiden Zeilen eines produktivreifen Handlers bewusst langweilig: prüfen, ob das Rechteck groß genug ist, um überhaupt etwas zu zeichnen, und es dann dem Geometrie-Helfer übergeben. Ein Control, das mit Breite null angeordnet oder bei einem responsiven Umbruch

zusammengedrückt wird, kehrt einfach zurück. Kein Zeichenpfad ist so wichtig, dass er eine Exception auf dem Bildschirm einer Benutzerin rechtfertigt.

Formen, Pfade, Text und transformierte Inhalte

`Graphics` ist eine zustandsbehaftete, unmittelbare Zeichen-API, und die Familien, die das Gauge braucht, sind wenige. Konturen kommen von `DrawArc`, `DrawLine` und `DrawPolygon`; gefüllte Geometrie von `FillPie`, `FillEllipse` und `FillRectangle`; Text von `DrawString` mit einem `Font`, einem `Brush` und einem `StringFormat`; zusammengesetzte Formen und Kurven von einem `GraphicsPath` aus `System.Drawing.Drawing2D`. Clipping und Transformationen runden den Satz ab.

Bei den Transformationen laufen Handler meist aus dem Ruder. `TranslateTransform` und `RotateTransform` anzuwenden, um den Zeiger zu schwenken, ist einfach; sie vor dem Zeichnen der Beschriftung rückgängig zu machen, nicht. Öffnen Sie einen Grafikcontainer mit `BeginContainer()`, zeichnen Sie in einem `try` und setzen Sie `EndContainer(state)` in das `finally`: Die von Wisej.NET gelieferte verwaltete Zeichenimplementierung hat kein Paar `Graphics.Save / Restore`, und der Container ist dessen Entsprechung. Dieselbe Disziplin taucht in Modul 4 als Canvas-Paar `Save()` und `Restore()` wieder auf, das diese Namen tatsächlich trägt.

Text verdient dieselbe Sorgfalt: `MeasureString` und die Ausrichtung über `StringFormat` überstehen eine verschobene Beschriftung, ein fest verdrahteter Versatz nicht. Für Bilder gilt eine zusätzliche Regel, denn `Paint` übernimmt den Hintergrund. Ein Wasserzeichen wird ausdrücklich mit `DrawImage` gezeichnet, aus einem einmal dekodierten Bild, das dem Control gehört, und nie aus einer Datei, die im Handler geöffnet wird.

Ein wiederverwendbares Control, dessen Properties richtig invalidieren

Im Lab wird `TelemetryGauge` zu einem echten Control statt zu einem an ein Panel geschraubten Handler. Das Rendering lebt in der Klasse, und Verwender setzen fachliche Properties -- `Value`, `Minimum`, `Maximum`, `WarningThreshold`, `CriticalThreshold`. Wisej.NET stellt an `Control` außerdem einen geschützten Hook `OnWebPaint(Graphics)` für Bibliotheksautoren bereit, während das öffentliche `Paint`-Event der einfachste Kompositionsmechanismus bleibt.

Jeder Setter folgt einem Rezept: auf einen zulässigen Bereich begrenzen, mit dem Gespeicherten vergleichen, zurückkehren, wenn sich nichts geändert hat, zuweisen und dann einmal invalidieren. Ein Wert außerhalb des Bereichs wird begrenzt statt abgelehnt, und ein umgekehrter Bereich wird normalisiert, statt einen negativen Winkelbereich erzeugen zu dürfen. Danach darf der Zeichencode davon ausgehen, dass seine Eingaben vernünftig sind.

```

public float Value
{
    get => _value;
    set
    {
        var low = Math.Min(Minimum, Maximum); // an inverted range is
        var high = Math.Max(Minimum, Maximum); // normalised, not thrown on
        var next = Math.Clamp(value, low, high);
        if (Math.Abs(_value - next) < 0.001f)
            return; // nothing visual changed
        _value = next;
        AccessibleDescription = $"{Caption}: {_value:0.#}";
        Invalidate(); // exactly once per real change
    }
}

```

Die zweite Hälfte des Controls ist `GaugeGeometry` : ein Helfer, der ein Rechteck und den normalisierten Wert entgegennimmt und Ringgrenzen, einen Startwinkel, einen Winkelbereich, die Zeigerpunkte und das Beschriftungsrechteck zurückgibt. Er fasst kein `Graphics` an, sodass Sie seine Ausgabe in einem schlichten Test prüfen können -- und dasselbe Gauge in Modul 6 off screen zu rendern kostet dann fast nichts mehr.

Wem welches freizugebende Objekt gehört

Der Besitz ist es, der ein gezeichnetes Control auf einem gemeinsam genutzten Server aus Schwierigkeiten heraushält. Der Handler besitzt, was er erzeugt -- Pens, einfarbige Brushes und Verlaufs-Brushes, Fonts, Paths, jede Bitmap -- und eine `using` -Deklaration gibt jedes davon am Ende der Methode frei. Er besitzt `e.Graphics` nicht, und genau das freizugeben ist der klassische Fehler.

Zwei verwandte Gewohnheiten zählen genauso viel. Statische Framework-Objekte wie `Pens.Red` und `Brushes.Black` werden gemeinsam genutzt und müssen in Ruhe gelassen werden. Und ein veränderlicher Pen, der in einem `static` -Feld geparkt ist, ist schlimmer als ein Leck: Ein Wisej.NET-Server bedient viele Sitzungen und rendert womöglich für mehrere gleichzeitig, also bleibt der Rendering-Zustand lokal, sofern Sie nicht bewusst einen unveränderlichen, threadsicheren Cache entwerfen. Der Zeichenpfad öffnet außerdem nie eine Datenbankverbindung, ruft keinen Service auf und startet keinen Timer.

Das Kostenbudget und der Messwert, den niemand sieht

Ein gezeichnetes Control gibt an drei Stellen aus: Server-CPU beim Zeichnen, Serverspeicher für das temporäre Bild sowie Netzwerk plus Browser-Dekodierung für das gesendete Bild. Eine Statusleiste und eine fensterfüllende Heatmap sind für den Browser beide nur Bilder, und ihre Kosten liegen nicht annähernd beieinander – messen Sie also alle drei, wenn eine Fläche wächst. Genau das hält `PaintCost.md` fest.

Die Invalidierungsstrategie ist die andere Hälfte des Budgets. Schreiben Sie die visuellen Abhängigkeiten auf -- typischerweise Wert, Beschriftung, Farbpalette und Größe -- und zeichnen Sie nur dafür neu. Bewegt sich ein Wert innerhalb einer Anfrage zehnmal, aktualisieren Sie den Zustand zehnmal, arbeiten Sie aber auf eine einzige visuelle Aktualisierung am Ende hin, und lassen Sie nie einen serverseitigen Timer laufen,

nur um ein Bild zu animieren. Wenn die Zahlen die Pixel nicht mehr rechtfertigen, lautet die ehrliche Antwort: ein gethemtes Control, eine HTML-Darstellung oder eine Diagramm-Erweiterung.

Und schließlich: Eine gezeichnete Zahl ist keine barrierefreie Zahl. Wer auf assistive Technologien angewiesen ist, bekommt ein Bild, also veröffentlicht das Gauge seinen Messwert über

`AccessibleDescription`, und die Seite behält neben der Grafik ein textliches Äquivalent. Ergänzen Sie einen Rückfall für eine fehlende Schrift oder ein fehlendes Bild und einen minimalen Diagnosezustand statt einer Exception – dann können Sie das Control gefahrlos an ein anderes Team übergeben.



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:

- ✓ Das Paint-Event behandeln und mit PaintEventArgs.Graphics und ClipRectangle arbeiten.
- ✓ Formen, Pfade, Text, Bilder, Farbverläufe und transformierte Inhalte mit System.Drawing zeichnen.
- ✓ Ein wiederverwendbares, selbst gezeichnetes Control bauen, dessen öffentliche Properties es korrekt invalidieren.
- ✓ Die Regeln zur Lebensdauer von Pens, Brushes, Fonts, Bildern und Grafikobjekten anwenden.
- ✓ Erkennen, wann Zeichenkosten, PNG-Übertragung oder Barrierefreiheit eine andere Technik vorziehen lassen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Ergänzen Sie VisualOperationsStudio um ein wiederverwendbares TelemetryGauge-Control, das einen Ring, farbige Schwellenzonen, einen Zeiger, einen Zahlenwert und eine Beschriftung zeichnet. Geben Sie ihm die Properties Value, Minimum, Maximum, WarningThreshold und CriticalThreshold, deren Setter die Eingabe begrenzen, bei unveränderten Werten sofort zurückkehren und Invalidate() einmal aufrufen. Legen Sie die Koordinatenmathematik in einen seiteneffektfreien GaugeGeometry-Helfer, der ein Rectangle entgegennimmt und Ringgrenzen, Winkelbereiche, Zeigerpunkte und das Beschriftungsrechteck zurückgibt; zeichnen Sie dann aus diesen Werten mit per using begrenzten Pens, Brushes und einem GraphicsPath und lassen Sie das übergebene Graphics unangetastet. Behandeln Sie ein Control der Größe null, einen umgekehrten Wertebereich und einen Wert außerhalb des Bereichs, ohne eine Exception auszulösen, und veröffentlichen Sie den Messwert über AccessibleDescription, damit er nicht nur als Pixel existiert.

Ausgangspunkt

Sie öffnen VisualOperationsStudio aus Modul 1, wo `VisualOperationsPage` bereits ein einziges `TelemetrySample`-Modell hält und dessen Wert auf vier Oberflächen zeigt, darunter ein `Panel` mit einem direkt eingebetteten `Paint`-Handler, der einen Balken zeichnet. In diesem Lab wird dieser eingebettete Handler ersetzt: Sie fügen dem Projekt eine Klasse `TelemetryGauge` hinzu, platzieren drei Instanzen auf der Seite und binden jede an einen Messwert desselben Modells.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie eine Klasse `TelemetryGauge` hinzu, die von einem `Wisej.NET-Control` abgeleitet ist, geben Sie ihr Hintergrundfelder sowie `Value`, `Minimum`, `Maximum`, `WarningThreshold`, `CriticalThreshold` und `Caption` und schreiben Sie einen Setter, der begrenzt, vergleicht, früh zurückkehrt und einmal `Invalidate()` aufruft -- übertragen Sie diese Form dann auf die übrigen.
- ✓ Schreiben Sie `GaugeGeometry` als seiteneffektfreien Helfer: eine Methode, die das zeichenbare `Rectangle` und den normalisierten Wert entgegennimmt und Ringgrenzen, Start- und Winkelbereich, die Zeigerpunkte und das Beschriftungsrechteck zurückgibt, ohne irgendwo in der Datei auf `Graphics` zu verweisen.
- ✓ Hängen Sie den `Paint`-Handler an (oder überschreiben Sie den Zeichen-Hook) und zeichnen Sie streng aus dem Geometrieergebnis: die inaktive Skala, die Warn- und die kritische Zone, den aktiven Bogen, den als `GraphicsPath` gebauten Zeiger und danach Zahlenwert und Beschriftung, vermessen mit `MeasureString` und einem `StringFormat`.
- ✓ Setzen Sie jeden erzeugten Pen, Brush, Font und Path in eine `using`-Deklaration, verwenden Sie statische Framework-Objekte wie `Brushes.Black` ohne sie freizugeben und klammern Sie die `RotateTransform` des Zeigers in `Graphics.BeginContainer()` und `EndContainer(state)`, damit die Beschriftung in untransformierten Koordinaten gezeichnet wird.

- ✓ Sichern Sie die Grenzfälle ab: sofort zurückkehren, wenn das Rechteck zum Zeichnen zu klein ist, einen umgekehrten Bereich normalisieren, einen Wert außerhalb davon begrenzen, `AccessibleDescription` aus dem Setter setzen; ändern Sie dann die Browsergröße, verändern Sie den Wert über einen Button und halten Sie Bildgröße und Häufigkeit des Neuzeichnens in `PaintCost.md` fest.

Geforderte Ergebnisse

- ✓ TelemetryGauge-Control mit begrenzenden Value-, Minimum-, Maximum- und Schwellen-Properties, die pro echter Änderung genau einmal `Invalidate()` aufrufen
- ✓ Seiteneffektfreier GaugeGeometry-Helfer, der Ringgrenzen, Winkelbereiche, Zeigerpunkte und das Beschriftungsrechteck liefert
- ✓ Paint-Handler, der jeden selbst erzeugten Pen, Brush, Font und Path freigibt und `e.Graphics` niemals freigibt
- ✓ Absicherungen für Größe null, umgekehrten Bereich und Werte außerhalb des Bereichs, dazu eine `AccessibleDescription` mit dem Messwert
- ✓ `PaintCost.md` mit Notizen zur Größe des neu gezeichneten Bereichs, zur Häufigkeit des Neuzeichnens und dazu, wann ein gethemtes Control günstiger wäre

Wiederholungsfragen

- ✓ Ein Kollege verlagert die Mathematik für Ring und Zeiger direkt in den `Paint`-Handler, weil es „nur ein paar Zeilen“ sind. Welche zwei Dinge lassen sich danach nicht mehr günstig tun, und auf welches davon baut Modul 6 auf?
- ✓ Ihr Gauge ist an einen Maschinendatensatz gebunden, der auch einen Bedienernamen und ein Wartungsdatum trägt. Sollte das Control invalidieren, wenn sich diese Felder ändern? Begründen Sie Ihre Antwort über visuelle Abhängigkeiten und nicht über das Modell.
- ✓ Dasselbe Gauge soll auf ein Dashboard, das sechzig davon zeigt und sich alle zwei Sekunden aktualisiert. Was würden Sie zuerst messen, und was würden Sie ändern, wenn die Messwerte schlecht ausfallen?



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.