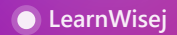


Listen, Bäume, Repeater und hierarchische Daten

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

ComboBox, ListBox, ListView, TreeView, DataRepeater und PropertyGrid; Virtual Mode der ListView, verzögertes Laden der TreeView, stabile Knotenidentität und auswahlgesteuertes Laden von Details. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

ComboBox, ListBox, ListView, TreeView, DataRepeater und PropertyGrid; Virtual Mode der ListView, verzögertes Laden der TreeView, stabile Knotenidentität und auswahlgesteuertes Laden von Details.

1. Das Problem im Produktivbetrieb

Listen- und Baum-Controls werden oft als einfache Auswahl-Widgets behandelt, doch in einer Geschäftsanwendung sind sie die Landkarte des Benutzers: Ordner, Kunden, Aufträge, Verfahren, Rollen, Aufgaben, Dokumente, Produktkataloge und die Module der Anwendung selbst. Der Dokument-Explorer der Operations Console, den Sie in diesem Modul bauen, muss schnell bleiben, wenn der Kategoriebaum tief und die Dokumentliste lang ist, und jeder Klick muss auf einen echten Geschäftsdatensatz verweisen statt auf ein Stück Anzeigetext.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Datenform	Die erste Entwurfsfrage: Sind die Daten flach, gruppiert, hierarchisch, als Karten wiederholt oder eine Menge von Properties? Die Antwort bestimmt das Control.
ListView-Ansichten	Ein Control, das dieselbe Sammlung als LargeIcon, SmallIcon, Details oder Tile anzeigt; die natürliche Wahl für dateiähnliche und katalogartige Bildschirme.
Virtual Mode	Ein ListView-Modus, in dem die Anwendung <code>VirtualListSize</code> setzt und <code>RetrieveVirtualItem</code> beantwortet, sodass Einträge nur für die Zeilen existieren, nach denen das Control fragt.
verzögertes Laden	Beim Start nur die oberen Ebenen eines Baums laden und Kinder erst beim Aufklappen eines Knotens holen, damit der Start günstig und die Oberfläche reaktionsschnell bleibt.
virtuelles Scrollen und Vorabladen	Im Browser nur die sichtbaren Zeilen einer großen Liste oder eines großen Baums rendern, wobei <code>PrefetchItems</code> einige zusätzliche Zeilen vorab rendert, damit das Scrollen flüssig bleibt.
stabile Identität	Eine eindeutige fachliche ID in <code>Tag</code> oder <code>Name</code> des Knotens oder Eintrags oder in einer ViewModel-Zuordnung, niemals aus dem angezeigten Text abgeleitet.
Auswahl als Event	Ein Auswahlwechsel ermittelt eine ID und fordert die Daten bei einem Service an; im Code, der Knoten oder Einträge aufbaut, läuft niemals eine Abfrage.

3. Der rote Faden der Lektion

Listen und Bäume sind die Landkarte des Benutzers

Jedes Listen-Control in Wisej.NET beantwortet eine andere Frage. `ComboBox` wählt auf kompaktem Raum einen Wert aus; ihr `DropDownStyle` entscheidet, ob der Benutzer einen freien Wert eintippen kann oder aus der Liste wählen muss. Sie passt zu kleinen und mittelgroßen Referenzlisten und nicht zu Tausenden von Einträgen: Ein Dropdown mit Scrollbalken ist keine Suche. `ListBox` und `CheckedListBox` sind für den Fall gedacht, dass die Liste selbst sichtbar bleibt und das Auswählen oder Ankreuzen die eigentliche Aufgabe ist. `ListView` ist für Sammlungen gedacht, die von einer Darstellungsform profitieren: Symbole, Kacheln oder Detailzeilen. `TreeView` ist für Hierarchie, Aufklappen und den Kontext von Eltern und Kindern da.

`DataRepeater` gehört dorthin, wo eine wiederholte visuelle Vorlage mehr aussagt als ein Grid: ein Task-Board, Benachrichtigungskarten, Produktkarten, Suchergebniskarten. Jedes wiederholte `ItemTemplate` ist eine Menge von Server-Controls mit Browser-Widgets dahinter, daher braucht eine große Ergebnismenge seitenweises Laden, Filter oder eine Virtualisierungsstrategie, bevor sie eine hübschere Karte braucht.

`PropertyGrid` ist der generische Inspektor: Richten Sie `SelectedObject` auf ein Objekt, und der Benutzer kann dessen Properties lesen oder bearbeiten. Das passt zu Administrationsbildschirmen, Entwicklerwerkzeugen und Objektinspektoren und weniger zu ausgefeilter Dateneingabe für Endbenutzer, bei der die generische Struktur Implementierungsdetails offenlegt.

Die Entscheidungsregeln passen auf eine Karteikarte: ComboBox für kompakte Auswahl, ListView, wenn die Darstellungsform zählt, TreeView, wenn die Hierarchie zählt, DataRepeater, wenn jeder Datensatz eine Karte braucht, PropertyGrid für generische Inspektion, Virtual Mode oder verzögertes Laden, bevor eine Liste oder ein Baum langsam wird, und stabile IDs getrennt vom angezeigten Text. In der Operations Console sind die Kategorien eine `TreeView`, die Dokumente einer Kategorie eine `ListView` in der Details-Ansicht, und das ausgewählte Dokument öffnet sich in einem Detail- `UserControl`.

ListView-Ansichten und Virtual Mode

Eine `ListView` ändert ihre Darstellung über die Property `View`, sodass ein Dokumentkatalog zwischen Kacheln mit großen Symbolen und einer Details-Ansicht mit Spalten wechseln kann, ohne etwas neu aufzubauen. Bei einer Handvoll Einträgen ist es in Ordnung, `ListViewItem`-Objekte direkt hinzuzufügen. Bei datengestützten Bildschirmen binden Sie das Control oder geben ihm eine klar benannte Lademethode, damit es genau eine Stelle gibt, an der aus Dokumenten Einträge werden.

Bei großen Sammlungen schalten Sie den Virtual Mode ein. Die `Items`-Collection ist dann nicht mehr der primäre Speicher: Die Anwendung teilt dem Control über `VirtualListSize` mit, wie viele Zeilen es gibt, und behandelt `RetrieveVirtualItem`, wobei sie ein `ListViewItem` erst erzeugt, wenn das Control nach einem bestimmten `ItemIndex` fragt. Der Handler muss günstig sein, daher liest er aus einer serverseitigen Seite oder einem Cache, den der Service beim Kategoriewechsel befüllt hat, niemals Zeile für Zeile aus der Datenbank. Image-Key und stabile ID werden im selben Moment am Eintrag gesetzt.

```
private void categoryTree_AfterExpand(object sender, TreeViewEventArgs e)
{
    if (e.Node.Nodes.Count != 1 || e.Node.Nodes[0].Name != "placeholder") return;
    e.Node.Nodes.Clear();
    foreach (var child in _documents.GetChildren((string)e.Node.Tag))
        e.Node.Nodes.Add(new TreeNode(child.Title) { Name = child.Id, Tag = child.Id,
        ImageKey = "folder" });
}

private void documentList_RetrieveVirtualItem(object sender, RetrieveVirtualItemEventArgs e)
{
    var doc = _page[e.ItemIndex]; // serverseitige Seite, befüllt von DocumentService
    e.Item = new ListViewItem(doc.Title) { Name = doc.Id, Tag = doc.Id, ImageKey = doc.Kind
};
}
```

TreeView: verzögertes Laden, Kontrollkästchen, Bilder und große Bäume

Eine `TreeView` zeigt `TreeNode`-Objekte in einer Hierarchie. Einen tiefen Baum beim Start vollständig zu laden ist der häufigste Performance-Fehler in dieser Familie, daher verwendet das Lab verzögertes Laden: Bauen Sie nur die Kategorien der obersten Ebene auf, geben Sie jeder ein einzelnes Platzhalter-Kind, damit das Aufklapp-Symbol erscheint, und ersetzen Sie den Platzhalter in `AfterExpand` durch die echten Kinder aus dem Service. Im obigen Codeausschnitt ist der `Name` des Platzhalters die Markierung, an der der Handler erkennt, ob der Knoten bereits geladen wurde.

Zwei Properties verändern, was der Baum für den Benutzer bedeutet. `CheckBoxes` zeigt neben jedem Knoten ein Kontrollkästchen an und macht den Baum zu einer Fläche mit Mehrfachauswahl für

Stapelvorgänge, etwa das Archivieren mehrerer Kategorien. Knotenbilder, zugewiesen über eine `ImageList` und den `ImageKey` jedes Knotens, lassen den Benutzer Ordner, Dokumente, Warnungen und Endknoten auf einen Blick unterscheiden; die `ListView` verwendet dieselben Image-Keys, sodass beide Controls dieselbe Bildsprache sprechen.

Ist ein Baum auch nach dem verzögerten Laden groß, senkt virtuelles Scrollen den Rendering-Aufwand im Browser, weil nur die sichtbaren Knoten gerendert werden, und `PrefetchItems` rendert einige zusätzliche Zeilen außerhalb des sichtbaren Bereichs vorab, damit das Scrollen flüssig bleibt. Keines von beiden ersetzt das verzögerte Laden; sie sind die zweite und dritte Strategie, in dieser Reihenfolge.

Hierarchische Daten brauchen außerdem eine klare Identität. Dieselbe Beschriftung taucht unter vielen Zweigen auf: Einen Ordner „Documents“ gibt es unter jedem Kunden, und zwei Kategorien können denselben Titel tragen. Speichern Sie eine stabile ID in `Tag` oder `Name` des Knotens oder in einem ViewModel-Dictionary mit dem Knoten als Schlüssel, damit der ausgewählte Knoten immer genau einem Geschäftsobjekt entspricht. Anzeigetext ist für Menschen, Identität für den Code.

Auswahl ist ein Event, keine Abfrage

Die Auswahl sollte ein Event sein, keine Datenbankabfrage, die im Rendering-Code vergraben ist. Wählt der Benutzer einen Knoten im Baum oder einen Eintrag in der Liste, erledigt der Handler für `AfterSelect` oder `SelectedIndexChanged` drei Dinge: Er ermittelt die stabile ID aus `Tag`, fordert die Daten bei einem `DocumentService` oder einem ViewModel an und übergibt das Ergebnis an das Detail-`UserControl`. Das Detail-Control stellt eine Methode bereit, die ein Dokumentmodell entgegennimmt, und weiß nichts über den Baum, die Liste oder die Datenbank.

Diese Aufteilung hält den Bildschirm vorhersehbar und testbar: Controls zeigen den Ansichtszustand, Services holen Geschäftsdaten, und die Seite koordiniert den Ablauf. Sie gibt dem Benutzer auch ehrliches Feedback: Die Seite kann einen Ladezustand anzeigen, einen fehlgeschlagenen Serviceaufruf abfangen und eine verständliche Meldung zeigen, ohne halb aufgebaute Einträge in der Liste zurückzulassen. Bevor Sie den Explorer für fertig erklären, gehen Sie die Checkliste für große Datenmengen durch:

- Keine unbegrenzten Ladevorgänge; Virtual Mode oder verzögertes Laden deckt jeden großen Pfad ab.
- Zwischen den Controls und der Datenbank liegt ein serverseitiger Cache oder eine Seite.
- Stabile IDs sind auf jedem Knoten und Eintrag vom Anzeigetext getrennt.
- Die Performance ist mit realistischen Daten getestet, und Sie wissen, was bei zehnfacher Menge zuerst bricht.



Typische Fehler

Einen ganzen tiefen Baum beim Start laden. Fachliche Identität nur im Knotentext speichern. Tausende reichhaltige wiederholte Controls in einem `DataRepeater` erzeugen, statt seitenweise zu laden oder zu virtualisieren. Eine `ComboBox` als schlechten Ersatz für eine Suche verwenden. Auswahl-Events direkt mit Datenbankcode verknüpfen, ohne kapselnden Service dazwischen.



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ ComboBox, ListBox, CheckedListBox, ListView, TreeView, DataRepeater und PropertyGrid für das jeweils passende Interaktionsmuster einsetzen.
- ✓ den Virtual Mode der ListView mit RetrieveVirtualItem und VirtualListSize umsetzen.
- ✓ das Laden der TreeView, Kontrollkästchen, Icons und Strategien für große Bäume mit virtuellem Scrollen und Vorabladen umsetzen.
- ✓ die Auswahl in einem Baum oder einer Liste mit einem Detail-Panel verbinden, ohne den Datenabruf an das Rendering der Oberfläche zu koppeln.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Erstellen Sie den Dokument-Explorer der Operations Console: eine TreeView mit Kategorien oder Ordnern, bei der an jedem Knoten eine stabile ID hängt, eine ListView mit den Dokumenten des ausgewählten Knotens, verzögertes Laden oder Virtual Mode auf mindestens einem Listen- oder Baumpfad, Icons oder Image-Keys für die verschiedenen Elementtypen und ein Detail-UserControl, das bei jedem Auswahlwechsel über eine Service- oder ViewModel-Schicht geladen wird.

Ausgangspunkt

Öffnen Sie die Projektmappe `OperationsConsole` aus Modul 3 und wechseln Sie zum Tab Lists and Trees der Shell: Er enthält noch die Platzhalterseite aus Modul 1. Legen Sie die Arbeit dieses Moduls in einen eigenen Ordner oder Namespace (zum Beispiel `Explorer`), damit das Abschlussprojekt sie wiederverwenden kann, und verwenden Sie einen kleinen In-Memory- `DocumentService` mit einigen hundert generierten Dokumenten, damit das Verhalten bei großen Datenmengen auch ohne Datenbank sichtbar wird.

Vorgeschlagenes Vorgehen

- ✓ Notieren Sie zuerst die Datenform: Kategorien sind hierarchisch, die Dokumente einer Kategorie eine flache Sammlung, und ein Dokument hat Properties. Wählen Sie `TreeView`, `ListView` und ein Detail- `UserControl` und halten Sie fest, warum Sie sich gegen `ComboBox`, `DataRepeater` oder `PropertyGrid` entschieden haben.
- ✓ Bauen Sie den Kategoriebaum verzögert auf: Fügen Sie nur die Kategorien der obersten Ebene hinzu, geben Sie jeder ein Platzhalter-Kind, speichern Sie die Kategorie-ID in `Tag` oder `Name` und ersetzen Sie den Platzhalter in `AfterExpand` über den Service durch die echten Kinder.
- ✓ Befüllen Sie die Dokument- `ListView` anhand des ausgewählten Knotens: Binden Sie entweder die vom Service gelieferte Seite oder aktivieren Sie den Virtual Mode, setzen Sie `VirtualListSize` und erzeugen Sie Einträge in `RetrieveVirtualItem` aus einer serverseitigen Seite, niemals über eine Abfrage pro Zeile.
- ✓ Fügen Sie eine `ImageList` hinzu, die beide Controls gemeinsam nutzen, und weisen Sie pro Elementtyp (Ordner, Vertrag, Rechnung, Zeichnung, Warnung) einen `ImageKey` zu, damit Baum und Liste dieselbe Bildsprache sprechen.
- ✓ Erstellen Sie `DocumentDetailControl` mit einer öffentlichen Methode, die ein Dokumentmodell entgegennimmt; ermitteln Sie in den Auswahl-Handlern die stabile ID, rufen Sie den Service in einem try/catch auf, zeigen Sie einen Ladezustand und bei einem Fehler eine verständliche Meldung, und übergeben Sie das Ergebnis an das Detail-Control.

Geforderte Ergebnisse

- ✓ TreeView mit Kategorien und einer stabilen ID an jedem Knoten

- ✓ ListView mit den Dokumenten des ausgewählten Baumknotens
- ✓ Verzögertes Laden oder Virtual Mode auf mindestens einem Listen- oder Baumpfad
- ✓ Detail-UserControl, das bei der Auswahl über eine Service- oder ViewModel-Schicht geladen wird
- ✓ Icons oder Image-Keys für die verschiedenen Elementtypen

Wiederholungsfragen

- ✓ Zwei Kategorien unter verschiedenen Kunden heißen beide „Contracts“. Was genau verwendet Ihr Auswahl-Handler, um sie zu unterscheiden, und wo liegt dieser Wert?
- ✓ Wenn die Anzahl der Dokumente in einer Kategorie von 300 auf 300.000 wüchse: Welcher Teil Ihres Explorers würde zuerst brechen, und welche Property oder welches Event würden Sie zur Behebung ändern?
- ✓ Welche Zeilen Ihres Auswahl-Handlers ließen sich ohne Browser in einen Unit-Test verschieben, und welche Zeilen gehören stattdessen in das Detail-UserControl?



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.