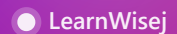


Listes, arbres, répéteurs et données hiérarchiques

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



Guide de la leçon

ComboBox, ListBox, ListView, TreeView, DataRepeater et PropertyGrid ; le mode virtuel de ListView, le chargement différé de TreeView, une identité de nœud stable et le chargement du détail piloté par la sélection. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Objectif en production

ComboBox, ListBox, ListView, TreeView, DataRepeater et PropertyGrid ; le mode virtuel de ListView, le chargement différé de TreeView, une identité de nœud stable et le chargement du détail piloté par la sélection.

1. Le problème en production

Les contrôles de liste et d'arbre sont souvent considérés comme de simples widgets de sélection, mais dans une application de gestion ils sont la carte de l'utilisateur : dossiers, clients, commandes, procédures, rôles, tâches, documents, catalogues de produits et modules de l'application elle-même. L'explorateur de documents d'Operations Console que vous construisez dans ce module doit rester rapide quand l'arbre des catégories est profond et la liste des documents longue, et chaque clic doit correspondre à un véritable enregistrement métier plutôt qu'à un morceau de texte affiché.

2. Concepts clés

Concept	Pourquoi c'est important
forme des données	La première question de conception : les données sont-elles plates, groupées, hiérarchiques, répétées en cartes, ou un ensemble de propriétés ? La réponse détermine le contrôle.
vues du ListView	Un seul contrôle qui affiche la même collection en LargeIcon, SmallIcon, Details ou Tile : le choix naturel pour les écrans de type fichiers ou catalogue.
mode virtuel	Un mode du ListView dans lequel l'application définit <code>VirtualListSize</code> et répond à <code>RetrieveVirtualItem</code> , si bien que les éléments n'existent que pour les lignes que le contrôle demande.
chargement différé	Ne charger que les niveaux supérieurs d'un arbre au démarrage et récupérer les enfants au déploiement d'un nœud, pour que le démarrage reste peu coûteux et l'interface réactive.
défilement virtuel et préchargement	N'afficher dans le navigateur que les lignes visibles d'une grande liste ou d'un grand arbre, avec <code>PrefetchItems</code> qui pré-affiche quelques lignes supplémentaires pour un défilement fluide.
identité stable	Un ID métier unique stocké dans le <code>Tag</code> ou le <code>Name</code> du nœud ou de l'élément, ou dans une table de correspondance du view model, jamais déduit du texte affiché.
la sélection comme événement	Un changement de sélection résout un ID et demande les données à un service ; aucune requête ne s'exécute jamais dans le code qui construit les nœuds ou les éléments.

3. Le fil de la leçon

Les listes et les arbres sont la carte de l'utilisateur

Chaque contrôle de liste de Wisej.NET répond à une question différente. `ComboBox` sélectionne une valeur dans un espace compact ; son `DropDownStyle` détermine si l'utilisateur peut saisir une valeur libre ou doit choisir dans la liste. Il convient aux listes de référence petites et moyennes et pas aux milliers d'éléments : une liste déroulante avec barre de défilement n'est pas une recherche. `ListBox` et `CheckedListBox` servent quand la liste elle-même reste visible et que sélectionner, ou cocher, est la tâche. `ListView` sert aux collections qui gagnent à un style de présentation : icônes, mosaïque ou lignes de détail. `TreeView` sert à la hiérarchie, au déploiement et au contexte parent/enfant.

`DataRepeater` a sa place là où un modèle visuel répété communique mieux qu'une grille : un tableau de tâches, des cartes de notification, des fiches produit, des cartes de résultats de recherche. Chaque `ItemTemplate` répété est un ensemble de contrôles serveur adossés à des widgets navigateur, donc un grand ensemble de résultats a besoin de pagination, de filtrage ou d'une stratégie de virtualisation avant d'avoir besoin d'une plus jolie carte. `PropertyGrid` est l'inspecteur générique : pointez `SelectedObject` sur un objet et l'utilisateur peut lire ou modifier ses propriétés. Cela convient aux écrans d'administration, aux outils de développement et aux inspecteurs d'objets, et moins à une saisie soignée destinée aux utilisateurs finaux, où la structure générique expose des détails d'implémentation.

Les règles de décision tiennent sur une fiche : ComboBox pour une sélection compacte, ListView quand le style de présentation compte, TreeView quand la hiérarchie compte, DataRepeater quand chaque enregistrement a besoin d'une carte, PropertyGrid pour l'inspection générique, mode virtuel ou chargement différé avant qu'une liste ou un arbre ne devienne lent, et des ID stables stockés séparément du texte affiché. Dans Operations Console, les catégories sont un `TreeView`, les documents d'une catégorie un `ListView` en vue Details, et le document sélectionné s'ouvre dans un `UserControl` de détail.

Vues du ListView et mode virtuel

Un `ListView` change de présentation via sa propriété `View`, si bien qu'un catalogue de documents peut passer d'une mosaïque à grandes icônes à une vue Details en colonnes sans rien reconstruire. Pour une poignée d'éléments, ajouter directement des objets `ListViewItem` convient. Pour les écrans alimentés par des données, liez le contrôle ou donnez-lui une méthode de chargement clairement nommée, pour qu'il existe un seul endroit où les documents deviennent des éléments.

Pour les grandes collections, activez le mode virtuel. La collection `Items` n'est alors plus le stockage principal : l'application indique au contrôle combien de lignes existent via `VirtualListSize` et gère `RetrieveVirtualItem`, en ne créant un `ListViewItem` que lorsque le contrôle le demande pour un `ItemIndex` donné. Le gestionnaire doit être peu coûteux : il lit dans une page ou un cache côté serveur que le service a rempli au changement de catégorie, jamais dans la base ligne par ligne. La clé d'image et l'ID stable sont définis sur l'élément au même moment.

```
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]; // server-side page filled by DocumentService
    e.Item = new ListViewItem(doc.Title) { Name = doc.Id, Tag = doc.Id, ImageKey = doc.Kind
};
}
```

TreeView : chargement différé, cases à cocher, images et grands arbres

Un `TreeView` affiche des objets `TreeNode` dans une hiérarchie. Charger entièrement un arbre profond au démarrage est l'erreur de performance la plus courante de cette famille, c'est pourquoi le lab utilise le chargement différé : ne construisez que les catégories de premier niveau, donnez à chacune un unique enfant provisoire pour que l'icône de déploiement apparaisse, et dans `AfterExpand` remplacez cet enfant provisoire par les vrais enfants récupérés auprès du service. Dans l'extrait ci-dessus, le `Name` de l'enfant provisoire est le marqueur qui indique au gestionnaire si le nœud a déjà été chargé.

Deux propriétés changent ce que l'arbre signifie pour l'utilisateur. `CheckBoxes` affiche une case à cocher à côté de chaque nœud et transforme l'arbre en surface de multisélection pour des opérations groupées comme l'archivage de plusieurs catégories. Les images de nœud, attribuées via une `ImageList` et l'`ImageKey` de chaque nœud, permettent à l'utilisateur de distinguer d'un coup d'œil dossiers, documents, avertissements et nœuds terminaux ; le `ListView` réutilise les mêmes clés d'image pour que les deux contrôles parlent le même langage visuel.

Quand un arbre reste volumineux même avec le chargement différé, le défilement virtuel réduit le coût de rendu dans le navigateur en n'affichant que les nœuds visibles, et `PrefetchItems` pré-affiche quelques lignes supplémentaires hors de la zone visible pour que le défilement reste fluide. Aucun des deux ne remplace le chargement différé ; ce sont les deuxième et troisième stratégies, dans cet ordre.

Les données hiérarchiques ont aussi besoin d'une identité claire. Le même libellé apparaît sous de nombreuses branches : un dossier « Documents » existe sous chaque client, et deux catégories peuvent porter le même titre. Stockez un ID stable dans le `Tag` ou le `Name` du nœud, ou dans un dictionnaire du view model indexé par nœud, pour que le nœud sélectionné corresponde toujours à un unique objet métier. Le texte affiché est destiné aux personnes ; l'identité est destinée au code.

La sélection est un événement, pas une requête

La sélection doit être un événement, pas une requête en base enfouie dans le code de rendu. Quand l'utilisateur sélectionne un nœud dans l'arbre ou un élément dans la liste, le gestionnaire de `AfterSelect` ou de `SelectedIndexChanged` fait trois choses : résoudre l'ID stable depuis `Tag`, demander les données à un `DocumentService` ou à un view model, et transmettre le résultat au `UserControl` de détail. Le contrôle de détail expose une seule méthode qui accepte un modèle de document et ne sait rien de l'arbre, de la liste ni de la base de données.

Cette séparation garde l'écran prévisible et testable : les contrôles affichent l'état de la vue, les services récupèrent les données métier, la page coordonne le déroulement. Elle donne aussi à l'utilisateur un retour honnête : la page peut afficher un état de chargement, intercepter un appel de service en échec et afficher un message compréhensible sans laisser d'éléments à moitié construits dans la liste. Avant de considérer l'explorateur comme terminé, passez la liste de contrôle des gros volumes :

- Aucun chargement de données non borné ; le mode virtuel ou le chargement différé couvre chaque chemin volumineux.
- Une page ou un cache côté serveur s'intercale entre les contrôles et la base de données.
- Les ID stables sont séparés du texte affiché sur chaque nœud et chaque élément.
- Les performances sont testées avec des données réalistes, et vous savez ce qui casse en premier avec un volume dix fois plus grand.



Pièges courants

Charger entièrement un arbre profond au démarrage. Stocker l'identité métier uniquement dans le texte des nœuds. Créer des milliers de contrôles riches répétés dans un `DataRepeater` au lieu de paginer ou de virtualiser. Utiliser une `ComboBox` comme piètre substitut à une recherche. Brancher les événements de sélection directement sur du code d'accès à la base, sans encapsulation par un service.



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Utiliser ComboBox, ListBox, CheckedListBox, ListView, TreeView, DataRepeater et PropertyGrid pour le bon modèle d'interaction.
- ✓ Mettre en œuvre le mode virtuel du ListView avec RetrieveVirtualItem et VirtualListSize.
- ✓ Mettre en œuvre le chargement, les cases à cocher, les icônes du TreeView et les stratégies pour grands arbres avec défilement virtuel et préchargement.
- ✓ Relier la sélection dans un arbre ou une liste à un panneau de détail sans coupler la récupération des données au rendu de l'interface.

Guide du lab / de l'examen

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



Lab pratique

Créez l'explorateur de documents d'Operations Console : un `TreeView` de catégories ou de dossiers avec un ID stable attaché à chaque nœud, un `ListView` de documents lié au nœud sélectionné, un chargement différé ou un mode virtuel sur au moins un chemin de liste ou d'arbre, des icônes ou des clés d'image pour les différents types d'éléments, et un `UserControl` de détail chargé via une couche service ou view model à chaque changement de sélection.

Point de départ

Ouvrez la solution `OperationsConsole` du Module 3 et allez sur l'onglet Lists and Trees du shell : il contient encore la page provisoire du Module 1. Gardez le travail de ce module dans son propre dossier ou namespace (par exemple `Explorer`) pour que le projet de synthèse puisse le réutiliser, et utilisez un petit `DocumentService` en mémoire avec quelques centaines de documents générés pour que le comportement avec de gros volumes soit visible sans base de données.

Approche suggérée

- ✓ Notez d'abord la forme des données : les catégories sont hiérarchiques, les documents d'une catégorie forment une collection plate, et un document possède des propriétés. Choisissez `TreeView`, `ListView` et un `UserControl` de détail, et consignez pourquoi une `ComboBox`, un `DataRepeater` ou un `PropertyGrid` n'ont pas été retenus.
- ✓ Construisez l'arbre des catégories en chargement différé : n'ajoutez que les catégories de premier niveau, donnez à chacune un enfant provisoire, stockez l'ID de la catégorie dans `Tag` ou `Name`, et remplacez l'enfant provisoire par les vrais enfants dans `AfterExpand` via le service.
- ✓ Remplissez le `ListView` des documents à partir du nœud sélectionné : liez la page renvoyée par le service, ou activez le mode virtuel, définissez `VirtualListSize` et créez les éléments dans `RetrieveVirtualItem` à partir d'une page côté serveur, jamais d'une requête par ligne.
- ✓ Ajoutez une `ImageList` partagée par les deux contrôles et attribuez une `ImageKey` par type d'élément (dossier, contrat, facture, plan, avertissement), pour que l'arbre et la liste parlent le même langage visuel.
- ✓ Créez `DocumentDetailControl` avec une seule méthode publique qui accepte un modèle de document ; dans les gestionnaires de sélection, résolvez l'ID stable, appelez le service dans un try/catch, affichez un état de chargement et un message compréhensible en cas d'échec, et transmettez le résultat au contrôle de détail.

Livrables attendus

- ✓ `TreeView` de catégories avec un ID stable stocké sur chaque nœud
- ✓ `ListView` de documents lié au nœud d'arbre sélectionné
- ✓ Chargement différé ou mode virtuel sur au moins un chemin de liste ou d'arbre

- ✓ UserControl de détail chargé via une couche service ou view model lors de la sélection
- ✓ Icônes ou clés d'image pour les différents types d'éléments

Questions de révision

- ✓ Deux catégories rattachées à des clients différents s'appellent toutes deux « Contracts ». Qu'utilise exactement votre gestionnaire de sélection pour les distinguer, et où se trouve cette valeur ?
- ✓ Si le nombre de documents d'une catégorie passait de 300 à 300 000, quelle partie de votre explorateur casserait en premier, et quelle propriété ou quel événement modifieriez-vous pour y remédier ?
- ✓ Quelles lignes de votre gestionnaire de sélection pourraient passer dans un test unitaire sans navigateur, et lesquelles relèvent plutôt du UserControl de détail ?



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.