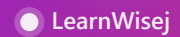


Appels client-serveur, sérialisation et accès aux objets

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



Guide de la leçon

Faire circuler données et commandes en toute sécurité entre les composants .NET et les widgets du navigateur. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



Objectif production

Faire circuler données et commandes en toute sécurité entre les composants .NET et les widgets du navigateur.

1. Le problème d'intégration

Une intégration utile ne se limite pas au rendu initial. Le serveur doit appeler des méthodes client, le client peut avoir besoin d'appeler des méthodes serveur, et les deux côtés ont besoin de contrats de données prévisibles.

2. Concepts clés

Concept	Comment l'utiliser
Call / CallAsync	Invoker des méthodes du widget client depuis le code serveur.
DTO	Utiliser de petits objets de transfert de données pour des contrats sérialisés stables.

3. Leçon de production

Wisej.NET fournit une couche de communication qui rend les appels du serveur vers le client naturels depuis C#. Le serveur peut appeler une méthode client et poursuivre, ou attendre une valeur du navigateur quand le déroulement l'exige. Le client peut aussi appeler des méthodes serveur exposées avec `[WebMethod]`, en style callback ou en style promise/await lorsque c'est pris en charge. La sérialisation est un atout, mais aussi une responsabilité. Wisej.NET sait sérialiser des objets anonymes, des DTO et des valeurs primitives, mais cela ne signifie pas que chaque objet serveur doit être envoyé au navigateur. Une bonne intégration utilise de petits objets adaptés aux besoins du widget JavaScript. Elle évite de laisser fuiter vers les options client les modèles métier internes, les entités de base de données ou un état sensible pour la sécurité. Les composants nommés sont reflétés dans le modèle objet client. C'est pratique pour les démonstrations et le code applicatif maîtrisé, mais les wrappers d'intégration doivent éviter de dépendre de chemins globaux fragiles.

Call, CallAsync et EvalAsync

Les trois appels du serveur vers le client diffèrent sur un seul point : le serveur attend-il ou non. `Call` nomme une méthode client, passe des arguments sérialisés et rend la main immédiatement ; l'invocation est mise en file d'attente et envoyée avec la réponse suivante. C'est le bon choix pour les commandes dont le serveur n'attend pas de réponse, comme définir la valeur de la jauge ou réinitialiser une animation.

`CallAsync` nomme aussi une méthode client, mais renvoie une Task qui se termine avec la valeur de retour de la méthode une fois que le navigateur a répondu. `EvalAsync` est le même mécanisme pour une expression JavaScript arbitraire évaluée dans la page.

Attendre un résultat du client suspend le gestionnaire côté serveur jusqu'à la fin de l'aller-retour avec le navigateur. C'est acceptable quand le déroulement dépend réellement de la valeur, par exemple lire la taille rendue avant de choisir une mise en page, et inutile quand la valeur est seulement journalisée. La règle du lab est d'utiliser `Call` par défaut et de ne recourir à une variante avec attente que lorsque l'instruction serveur suivante ne peut pas s'exécuter sans la réponse.

- Le résultat client est-il utilisé dans le même gestionnaire ; sinon, utilisez `Call`.
- Que se passe-t-il si le navigateur ne répond jamais, par exemple parce que l'onglet a été fermé.
- La valeur attendue est-elle validée avant qu'on lui fasse confiance.

Sérialisation et camelCase

Chaque argument passé au client et chaque valeur qu'il renvoie passent par le sérialiseur JSON de Wisej.NET. Valeurs primitives, tableaux, dictionnaires, objets anonymes et classes ordinaires sont tous sérialisés, et les noms de propriétés sont convertis en camelCase à la sortie : un `SelectedIndex` C# arrive en JavaScript sous la forme `selectedIndex`. Un code client qui lit le nom en PascalCase obtient undefined, sans aucune erreur.

Le sens inverse a ses propres règles. Un objet JavaScript renvoyé par `CallAsync` arrive côté serveur comme une valeur dynamique, et le code C# doit le lire avec les noms JavaScript ou le convertir en DTO typé. Les dates, les nombres qui dépassent la plage des entiers sûrs de JavaScript et les énumérations méritent un examen dans chaque contrat, car ce sont les valeurs le plus souvent altérées en transit.

La sérialisation est aussi l'endroit où les fuites de données se produisent. Une entité dotée d'une propriété de navigation vers une fiche client sérialise aussi ce client. La bonne habitude est de ne jamais faire transiter un objet métier, mais uniquement un objet construit pour ce passage.

Accéder au modèle objet client

Wisej.NET expose son arbre de composants aux scripts de la page. `App.MainPage` est la page principale courante, les contrôles qui ont un nom sont accessibles comme propriétés de leur conteneur, et `widget(id)` renvoie n'importe quel composant à partir de son identifiant client. Dans une démonstration, un script peut ainsi facilement récupérer un bouton, lire une zone de texte ou trouver la jauge en parcourant l'arbre.

Un wrapper d'intégration ne doit pas dépendre de cet arbre. Un chemin comme « la jauge du troisième panneau de la page principale » casse dès qu'un développeur renomme un contrôle ou le déplace dans une group box. Le wrapper dispose déjà d'une référence à lui-même avec `this` et à son propre

conteneur, et tout ce dont il a besoin du serveur arrive par les options et les appels. Les commandes de la jauge du lab sont toutes invoquées directement sur le widget, jamais recherchées par chemin.

Réservez le modèle objet client aux scripts de niveau page maîtrisés par l'application, comme un gestionnaire de raccourcis clavier, et considérez toute utilisation à l'intérieur d'un wrapper comme un point à relever en revue de code.

Des DTO petits et stables

Un objet de transfert de données destiné au réseau est une classe comportant quelques propriétés primitives, nommées d'après ce dont le widget a besoin, et rien d'autre. Le lab renvoie l'état client sélectionné de la jauge sous la forme d'un tel objet : la valeur courante, le fait qu'elle dépasse ou non le seuil, ainsi que la largeur et la hauteur rendues. Quatre nombres et un booléen, sérialisés en quelques dizaines d'octets et lisibles dans une entrée de journal.

Les petits DTO sont stables parce qu'ils changent rarement. Ajouter une propriété est sans risque, puisque le client ignore ce qu'il ne connaît pas. En renommer ou en supprimer une est un changement incompatible qui doit être versionné, et un DTO de quatre propriétés se versionne en quelques minutes. Un DTO qui reflète une entité métier change chaque fois que l'entité change, et chaque changement est une occasion de casser le client sans bruit.

Écrivez les DTO en premier, avant les appels qui les transportent. Une fois fixées les formes qui transitent sur le réseau, le code serveur comme l'InitScript disposent de quelque chose de concret sur quoi être testés.



Réflexe de production

Gardez clairement séparées la propriété de l'état côté Wisej.NET et le comportement du widget client. Documentez le contrat client/serveur avant de vous appuyer sur l'intégration en production.



Erreur courante à éviter

Copier du JavaScript de démonstration dans les écrans sans documenter le cycle de vie, les ressources, les contrats d'événements, le nettoyage ni les frontières de sécurité.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Utiliser ``Call``, ``CallAsync`` et ``EvalAsync`` depuis le code serveur pour interagir avec le widget client.
- ✓ Comprendre la sérialisation JSON de Wisej.NET, y compris le passage en camelCase des noms de propriétés.
- ✓ Utiliser à bon escient le modèle objet client, comme ``App.MainPage``, les widgets nommés et ``widget(id)``.
- ✓ Définir de petits DTO stables au lieu de transmettre des objets serveur volumineux ou accidentels.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, la démarche recommandée, les livrables attendus et l'auto-vérification avant la remise.



Lab pratique

Ajoutez des commandes à la jauge réutilisable : définir la valeur, lire la taille rendue, réinitialiser l'animation et renvoyer au serveur une partie de l'état côté client.

Démarche recommandée

- ✓ Ouvrez le projet existant du cours d'intégration Wisej.NET et exécutez-le en local.
- ✓ Repérez le composant serveur, le widget client, le fichier de ressources ou le point de terminaison concernés par ce module.
- ✓ Faites une petite modification à la fois et testez-la dans le navigateur avant de passer à la suite.
- ✓ Gardez le JavaScript propre à la bibliothèque tierce dans l'adaptateur ou le wrapper au lieu de le disperser dans les pages.
- ✓ Mettez à jour l'interface pour que l'apprenant voie clairement le succès, l'échec, la progression ou les changements de communication.
- ✓ Prenez des captures d'écran ou des notes qui prouvent que l'intégration fonctionne et qui expliquent le choix de conception.

Livrables attendus

- ✓ Appels de méthodes du serveur vers le client
- ✓ Un exemple de ``CallAsync`` ou ``EvalAsync``
- ✓ DTO utilisé pour renvoyer l'état client

Auto-vérification avant la remise

- ✓ Quand le code serveur doit-il attendre (await) un résultat du client ?
- ✓ Pourquoi le camelCase est-il important lorsqu'on passe des objets à JavaScript ?
- ✓ Pourquoi les DTO doivent-ils rester petits ?