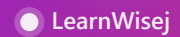


Événements, gestionnaires et contrats d'événements des widgets

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



Guide de la leçon

Transformer les callbacks de la bibliothèque tierce en événements .NET stables, sans laisser sa complexité se propager dans l'application. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



Objectif production

Transformer les callbacks de la bibliothèque tierce en événements .NET stables, sans laisser sa complexité se propager dans l'application.

1. Le problème d'intégration

Les widgets tiers produisent de nombreux événements dans le navigateur, mais la plupart des écrans applicatifs n'ont besoin que de quelques événements serveur significatifs, avec des charges utiles propres.

2. Concepts clés

Concept	Comment l'utiliser
fireWidgetEvent	Relayer au serveur les événements importants du widget.
Contrat de charge utile	Envoyer au code .NET des données d'événement réduites et significatives.

3. Leçon de production

L'intégration des événements n'est pas un exercice de relais mécanique. Une grille peut produire des dizaines d'événements navigateur. Un graphique peut générer des callbacks de survol, de clic, de sélection, de zoom, de rendu, d'export et de mise en page. Les envoyer tous au serveur crée du bruit et un trafic inutile. Un bon wrapper choisit les événements qui représentent des décisions métier ou des changements d'un état possédé par le serveur. Pour `Wisej.Web.Widget`, `fireWidgetEvent(name, data)` est le chemin direct vers le `WidgetEvent` côté serveur. Pour les contrôles personnalisés, `fireEvent` et `fireDataEvent` peuvent être traités via `OnWebEvent`, à condition que la configuration serveur câble l'événement. La différence compte, car les contrôles personnalisés peuvent partager des noms d'événements avec les événements intégrés et doivent transmettre les événements non traités à l'implémentation de base. La charge utile de l'événement doit être façonnée délibérément. Au lieu d'envoyer un objet événement tiers complet, envoyez `rowKey`, `fieldName`, `value`, `selection` ou un autre objet compact sur lequel le serveur peut agir directement. Le serveur ne devrait jamais avoir à fouiller une structure propre à la bibliothèque

tierce pour trouver la seule valeur qui l'intéresse, et la charge utile doit rester assez petite pour que son envoi à chaque interaction significative ne devienne pas un problème de performances.

Câbler les événements tiers au bon moment

Un callback tiers ne peut être attaché qu'une fois l'instance tierce créée ; l'endroit naturel est donc dans `init`, juste après la création, tant que `this` désigne encore le widget Wisej.NET. Certaines bibliothèques exposent une API d'événements sur l'instance, d'autres acceptent des callbacks dans l'objet d'options, et quelques-unes exigent de s'attacher aux éléments DOM qu'elles créent. Chaque approche fonctionne, à condition que le gestionnaire capture la référence au widget comme l'a enseigné le Module 2.

Les bibliothèques qui recréent leur instance lors de certains changements d'options exigent de rattacher les gestionnaires après chaque recréation ; c'est pourquoi le lab regroupe le câblage dans une seule fonction, appelée à la fois depuis `init` et depuis le chemin de recréation dans `update`. Attacher uniquement dans `init` laisserait la seconde instance muette.

Chaque attachement doit avoir son détachement dans le chemin de libération du widget. Un gestionnaire tiers qui survit à son widget retient ce widget en mémoire, et après suffisamment de changements d'écran, la fuite devient visible sous la forme d'un navigateur qui ralentit.

fireWidgetEvent vers le serveur

Pour un `Wisej.Web.Widget`, la méthode client `fireWidgetEvent(name, data)` est le seul chemin dont une intégration a besoin. Le nom est une courte chaîne choisie par l'intégration, comme `thresholdCrossed`, et les données sont n'importe quel objet sérialisable. Côté serveur, le `WidgetEvent` du widget se déclenche avec le nom et les données désérialisées, et le gestionnaire aiguille selon le nom pour décider quoi faire.

```
me.gauge.on('change', function (v) {
    if (v > options.threshold && !me.above) {
        me.above = true;
        me.fireWidgetEvent('thresholdCrossed', { value: v });
    }
});
```

L'exemple montre la discipline essentielle : le client décide quand un événement est significatif et ne le déclenche qu'une fois, au lieu de relayer chaque changement et de laisser le serveur filtrer. Une classe wrapper dérivée de `Wisej.Web.Widget` peut traiter `WidgetEvent` en interne et redéclencher un événement .NET typé comme `ThresholdExceeded`, si bien que le code applicatif ne voit jamais les noms sous forme de chaînes.

fireEvent et fireDataEvent pour les contrôles personnalisés

Un contrôle personnalisé construit sur le modèle de classes client déclenche ses événements différemment. `fireEvent(name)` déclenche un événement sans charge utile et `fireDataEvent(name, data)` un événement avec charge utile. Comme il s'agit d'événements client ordinaires, le serveur ne les reçoit que si la classe serveur du contrôle a déclaré son intérêt, ce que le lab décrit comme le câblage de l'événement dans la configuration serveur.

Côté serveur, les événements arrivent dans `OnWebEvent`, qui reçoit le nom et les données de l'événement. La redéfinition traite les noms qui lui appartiennent et doit transmettre tout le reste à l'implémentation de base, car un contrôle personnalisé partage le pipeline d'événements avec les événements intégrés que sa classe de base gère déjà, comme le focus et le redimensionnement. Avaler un événement inconnu casse un comportement qui n'a rien à voir avec l'intégration.

La différence entre les deux mécanismes est l'objet de l'auto-vérification du module. `WidgetEvent` est un événement fourre-tout unique sur un Widget ; `OnWebEvent` est le point d'entrée général des événements d'un contrôle, partagé avec le framework, et nécessite donc l'appel à la base.

Des charges utiles fiables pour le code .NET

La charge utile d'un événement est un contrat, et le lab la consigne dans un tableau : nom de l'événement, chaque champ avec son type, moment où l'événement se déclenche et événement .NET qu'il devient. Le `pointClicked` du graphique transporte un nom de série, une valeur x et une valeur y. Le `valueChanged` du bouton rotatif ne transporte que la nouvelle valeur. Aucune de ces charges utiles ne contient d'objet tiers, de nœud DOM ni de ligne de données entière.

Les petites charges utiles restent stables lors des mises à niveau de la bibliothèque tierce, car elles sont construites à partir de quelques valeurs extraites par le wrapper, et non d'après la forme de l'argument d'événement de la bibliothèque. Elles sont aussi sûres : le serveur considère chaque champ comme non fiable, le valide et recherche lui-même toute donnée sensible. Une clé de ligne dans une charge utile est une clé de recherche, jamais la preuve que l'utilisateur a le droit de modifier cette ligne.

- La charge utile ne contient-elle que des primitives et de petits objets.
- La même charge utile pourrait-elle être produite par une autre bibliothèque tierce.
- Chaque champ est-il validé côté serveur avant d'être utilisé.



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 :

- ✓ Câbler les événements tiers pendant l'initialisation ou après la création de l'instance.
- ✓ Utiliser `fireWidgetEvent` pour relayer les événements d'un `Wisej.Web.Widget`.
- ✓ Utiliser `fireEvent` et `fireDataEvent` avec des événements câblés pour les contrôles personnalisés.

- ✓ Concevoir des charges utiles d'événements petites, stables et significatives pour le code applicatif .NET.

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 événements côté serveur pour le franchissement du seuil de la jauge, le changement de valeur du bouton rotatif et le clic sur un point du graphique. Normalisez toutes les charges utiles d'événements en DTO simples.

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

- ✓ Trois gestionnaires d'événements côté client
- ✓ Trois gestionnaires d'événements côté serveur
- ✓ Tableau du contrat des charges utiles

Auto-vérification avant la remise

- ✓ Pourquoi ne pas relayer chaque événement de la bibliothèque tierce ?
- ✓ Quelle est la différence entre `WidgetEvent` et `OnWebEvent` ?
- ✓ Qu'est-ce qui rend stable la charge utile d'un événement ?