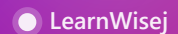


Events, Handler und Event-Verträge für Widgets

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Callbacks des Anbieters in stabile .NET-Events verwandeln, ohne dass die Komplexität des Anbieters in die Anwendung durchsickert. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.



Fokus: Produktivbetrieb

Callbacks des Anbieters in stabile .NET-Events verwandeln, ohne dass die Komplexität des Anbieters in die Anwendung durchsickert.

1. Das Integrationsproblem

Widgets von Drittanbietern erzeugen zahlreiche Browser-Events, doch die meisten Anwendungsmasken brauchen nur wenige aussagekräftige serverseitige Events mit sauberen Payloads.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
fireWidgetEvent	Wichtige Widget-Events an den Server weiterleiten.
Payload-Vertrag	Kleine, aussagekräftige Event-Daten an den .NET-Code senden.

3. Lektion aus der Praxis

Die Integration von Events ist keine mechanische Weiterleitungsübung. Ein Grid kann Dutzende Browser-Events erzeugen. Ein Diagramm kann Callbacks für Hover, Klick, Auswahl, Zoom, Rendering, Export und Layout auslösen. Sie alle an den Server zu senden erzeugt Rauschen und unnötigen Datenverkehr. Ein guter Wrapper wählt die Events aus, die fachliche Entscheidungen oder Änderungen am serverseitig verwalteten Zustand darstellen. Bei `Wisej.Web.Widget` ist `fireWidgetEvent(name, data)` der direkte Weg zum serverseitigen `WidgetEvent`. Bei eigenen Controls lassen sich `fireEvent` und `fireDataEvent` über `OnWebEvent` behandeln, sofern die Serverkonfiguration das Event verdrahtet. Der Unterschied ist wichtig, weil eigene Controls Event-Namen mit eingebauten Events teilen können und nicht behandelte Events an die Basisimplementierung weitergeben müssen. Der Event-Payload sollte bewusst gestaltet werden. Senden Sie statt eines vollständigen Event-Objekts des Anbieters `rowKey`, `fieldName`, `value`, `selection` oder ein anderes kompaktes Objekt, mit dem der Server direkt arbeiten kann. Der Server sollte sich nie durch eine anbieterspezifische Struktur wühlen müssen, um den einen Wert zu finden, der ihn interessiert, und der Payload sollte so klein bleiben, dass sein Auslösen bei jeder relevanten Interaktion nicht zum Performance-Problem wird.

Anbieter-Events zum richtigen Zeitpunkt verdrahten

Ein Callback des Anbieters lässt sich erst anhängen, wenn die Instanz des Anbieters existiert. Der natürliche Ort ist daher `init`, unmittelbar nach dem Erstellen, solange `this` noch auf das Wisej.NET-Widget verweist. Manche Bibliotheken bieten eine Event-API auf der Instanz, andere akzeptieren Callbacks im Optionsobjekt, und einige verlangen, dass Sie sich an DOM-Elemente hängen, die sie selbst erzeugen. Jeder dieser Wege funktioniert, sofern der Handler die Widget-Referenz so festhält, wie es Modul 2 gezeigt hat.

Bibliotheken, die ihre Instanz bei bestimmten Optionsänderungen neu erzeugen, brauchen die Handler nach jeder Neuerzeugung erneut. Deshalb hält das Lab die Verdrahtung in einer einzigen Funktion, die sowohl aus `init` als auch aus dem Neuerzeugungspfad in `update` aufgerufen wird. Würden Sie nur in `init` anhängen, bliebe die zweite Instanz stumm.

Zu jedem Anhängen gehört ein Abhängen im Dispose-Pfad des Widgets. Ein Handler des Anbieters, der sein Widget überlebt, hält das Widget im Speicher, und nach genügend Maskenwechseln macht sich das Leck als träger Browser bemerkbar.

fireWidgetEvent zum Server

Bei einem `Wisej.Web.Widget` ist die Client-Methode `fireWidgetEvent(name, data)` der einzige Weg, den eine Integration braucht. Der Name ist eine kurze, von der Integration gewählte Zeichenfolge wie `thresholdCrossed`, die Daten sind ein beliebiges serialisierbares Objekt. Auf dem Server wird das `WidgetEvent` des Widgets mit dem Namen und den deserialisierten Daten ausgelöst, und der Handler entscheidet anhand des Namens, was zu tun ist.

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

Das Beispiel zeigt die entscheidende Disziplin: Der Client entscheidet, wann ein Event aussagekräftig ist, und löst es einmal aus, statt jede Änderung weiterzuleiten und den Server filtern zu lassen. Eine von `Wisej.Web.Widget` abgeleitete Wrapper-Klasse kann `WidgetEvent` intern behandeln und ein typisiertes .NET-Event wie `ThresholdExceeded` erneut auslösen, sodass der Anwendungscode die Namen als Zeichenfolgen nie zu sehen bekommt.

fireEvent und fireDataEvent für eigene Controls

Ein eigenes Control auf Basis des Client-Klassenmodells löst Events anders aus. `fireEvent(name)` löst ein Event ohne Payload aus, `fireDataEvent(name, data)` eines mit Payload. Da es sich um gewöhnliche clientseitige Events handelt, erhält der Server sie nur, wenn die Serverklasse des Controls Interesse daran angemeldet hat – im Lab heißt das, das Event in der Serverkonfiguration zu verdrahten.

Auf dem Server kommen die Events in `OnWebEvent` an, das den Event-Namen und die Daten erhält. Die Überschreibung behandelt die Namen, für die sie zuständig ist, und muss alles andere an die Basisimplementierung weitergeben, denn ein eigenes Control teilt sich die Event-Pipeline mit den

eingebauten Events, die seine Basisklasse bereits behandelt, etwa focus und resize. Wer ein unbekanntes Event verschluckt, beschädigt Verhalten, das mit der Integration gar nichts zu tun hat.

Der Unterschied zwischen beiden Mechanismen ist der Selbstcheck dieses Moduls. `WidgetEvent` ist ein einziges Sammel-Event an einem Widget; `OnWebEvent` ist der allgemeine Event-Einstiegspunkt eines Controls, den es sich mit dem Framework teilt, und braucht deshalb den Aufruf der Basismethode.

Payloads, denen .NET-Code vertrauen kann

Ein Event-Payload ist ein Vertrag, und das Lab hält ihn in einer Tabelle fest: Event-Name, jedes Feld mit seinem Typ, wann das Event ausgelöst wird und zu welchem .NET-Event es wird. `pointClicked` des Diagramms transportiert einen Seriennamen, einen x-Wert und einen y-Wert. `valueChanged` des Knobs transportiert nur den neuen Wert. Keiner der beiden Payloads enthält ein Objekt des Anbieters, einen DOM-Knoten oder eine ganze Datenzeile.

Kleine Payloads bleiben über Updates des Anbieters hinweg stabil, weil sie aus wenigen Werten bestehen, die der Wrapper herauszieht, und nicht aus der Struktur der Event-Argumente des Anbieters. Sie sind außerdem sicher: Der Server behandelt jedes Feld als nicht vertrauenswürdig, validiert es und schlägt alles Sensible selbst nach. Ein Zeilenschlüssel im Payload ist ein Suchschlüssel – nie ein Beweis dafür, dass der Benutzer diese Zeile bearbeiten darf.

- Enthält der Payload nur primitive Werte und kleine Objekte?
- Könnte eine andere Anbieterbibliothek denselben Payload erzeugen?
- Wird jedes Feld auf dem Server validiert, bevor es verwendet wird?



Gewohnheit für den Produktivbetrieb

Halten Sie klar getrennt, welchen Zustand Wisej.NET besitzt und wie sich das Client-Widget verhält. Dokumentieren Sie den Client-Server-Vertrag, bevor Sie sich im Produktivbetrieb auf die Integration verlassen.



Typischer Fehler

Demo-JavaScript in Masken kopieren, ohne Lebenszyklus, Ressourcen, Event-Verträge, Aufräumen oder Sicherheitsgrenzen zu dokumentieren.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Events des Anbieters während der Initialisierung oder nach dem Erzeugen der Instanz verdrahten.
- ✓ `fireWidgetEvent`` zum Weiterleiten von Events bei `Wisej.Web.Widget` verwenden.
- ✓ `fireEvent`` und `fireDataEvent`` mit verdrahteten Events für eigene Controls verwenden.
- ✓ Event-Payloads entwerfen, die klein, stabil und für den .NET-Anwendungscode aussagekräftig sind.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab erstellen, der empfohlene Arbeitsablauf, die geforderten Ergebnisse und der Selbstcheck vor der Abgabe.



Praktisches Lab

Fügen Sie serverseitige Events für das Überschreiten eines Gauge-Schwellenwerts, die Wertänderung eines Knobs und den Klick auf einen Diagrammpunkt hinzu. Normalisieren Sie alle Event-Payloads in einfache DTOs.

Empfohlener Arbeitsablauf

- ✓ Öffnen Sie das vorhandene Projekt des Wisej.NET-Integrationskurses und starten Sie es lokal.
- ✓ Ermitteln Sie die Server-Komponente, das Client-Widget, die Ressourcendatei oder den Endpunkt, um die es in diesem Modul geht.
- ✓ Nehmen Sie jeweils nur eine kleine Änderung vor und testen Sie sie im Browser, bevor Sie weitermachen.
- ✓ Halten Sie anbieterspezifisches JavaScript im Adapter oder Wrapper, statt es über mehrere Seiten zu verstreuen.
- ✓ Passen Sie die UI so an, dass Erfolg, Fehler, Fortschritt oder Änderungen in der Kommunikation klar erkennbar sind.
- ✓ Halten Sie mit Screenshots oder Notizen fest, dass die Integration funktioniert, und begründen Sie die Designentscheidung.

Geforderte Ergebnisse

- ✓ Drei Event-Handler auf dem Client
- ✓ Drei serverseitige Event-Handler
- ✓ Tabelle der Payload-Verträge

Selbstcheck vor der Abgabe

- ✓ Warum sollten Sie nicht jedes Event des Anbieters weiterleiten?
- ✓ Worin unterscheiden sich ``WidgetEvent`` und ``OnWebEvent``?
- ✓ Was macht einen Event-Payload stabil?