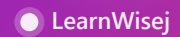


Postback, WebRequest, WebMethod und Datenendpunkte

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

JavaScript-Widgets, die ihre Daten per Ajax abrufen, mit Daten versorgen und die Anfragen dabei im Kontext der richtigen Wisej.NET-Komponente halten. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.



Fokus: Produktivbetrieb

JavaScript-Widgets, die ihre Daten per Ajax abrufen, mit Daten versorgen und die Anfragen dabei im Kontext der richtigen Wisej.NET-Komponente halten.

1. Das Integrationsproblem

Viele anspruchsvolle Widgets – vor allem Grids, Pivot-Grids, Reports, Viewer und Uploader – erwarten URLs oder Service-Endpunkte statt übertragener Daten.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
Postback-URL	Browseranfragen zurück an die richtige Komponenteninstanz leiten.
WebMethod	Aufrufe im RPC-Stil für Client-Callbacks oder Promises bereitstellen.

3. Lektion aus der Praxis

Manche Widgets lassen sich am einfachsten integrieren, wenn sie ihre Daten von einer URL abrufen. Das ist typisch für Daten-Grids, Pivot-Grids, Tabellenkalkulationen, Viewer, Uploader und Bibliotheken mit einer eigenen Datenquellen-Abstraktion. Wisej.NET stellt eine Postback-URL bereit, die an die Komponenteninstanz gebunden ist, sodass die Anfrage in der richtigen Session und im Kontext des richtigen Controls ankommt. Ein Postback-Handler verhält sich wie ein kleiner Webservice. Er erhält einen HttpRequest und schreibt eine HttpResponse. Die Antwort kann JSON für ein Grid sein, binärer Inhalt für eine Datei, Bilddaten für einen Viewer oder jedes andere Format, das die Anbieterbibliothek verlangt. Der Vorteil: Der Handler lebt bei der Komponenteninstanz und kann daher deren Zustand und die Services der Anwendung nutzen. `[WebMethod]` hat eine andere Form. Sie ist ideal, wenn der Browser eine öffentliche Servermethode aufrufen und einen gemarshalten Rückgabewert erhalten soll. Sie kann statisch sein oder an einem Container der obersten Ebene hängen. Für RPC-artige Aufrufe – einen Datensatz nachschlagen, einen Wert validieren oder eine kleine Ergebnismenge zurückgeben – ist sie meist die sauberere Lösung,

während Postback-Handler für Dateien, Bilder oder große Datenantworten, die eine Anbieterbibliothek von einer URL abrufen will, die bessere Wahl bleiben.

Eine URL an eine Komponenteninstanz leiten

Viele anspruchsvolle Widgets holen sich ihre Daten selbst. Ein Grid fordert eine Seite mit Zeilen an, ein Viewer streamt ein Dokument, ein Uploader sendet eine Datei. Sie alle wollen eine URL, und die Herausforderung besteht darin, dass diese URL die richtige Wisej.NET-Session und darin die richtige Komponenteninstanz erreichen muss. `GetPostBackURL()` auf dem Server bzw. `getPostbackUrl()` im Client-Widget liefert genau eine solche URL – codiert mit den Kennungen, die das Framework braucht, um die Anfrage an diese Komponente zuzustellen.

Der Wrapper übergibt diese URL dem Anbieter als Adresse der Datenquelle, meist in `init`, und kann eigene Query-Parameter anhängen, um Aktionen zu unterscheiden. Da die URL an eine Instanz gebunden ist, erhalten zwei Grids auf derselben Seite zwei unterschiedliche URLs, und jede Anfrage landet bei der Komponente, der die Daten gehören.

Behandeln Sie die URL als Session-gebunden und kurzlebig. Sie ist keine öffentliche API-Adresse, darf nicht gespeichert und nicht zwischen Benutzern geteilt werden.

Die Anfrage in WebRequest verarbeiten

Trifft eine Anfrage an der Postback-URL ein, wird das `WebRequest`-Event der Komponente mit dem rohen `HttpRequest` und der `HttpResponse` ausgelöst. Der Handler liest Aktion und Parameter, erzeugt die Daten, setzt den Content-Type und schreibt die Antwort. JSON für ein Grid, ein PDF-Stream für einen Viewer, ein Bild für ein Vorschaubild oder eine CSV-Datei für einen Export sind lediglich unterschiedliche Bodys mit unterschiedlichen Content-Types.

```
private void grid_WebRequest(object sender, WebRequestEventArgs e)
{
    var action = e.Request.QueryString["action"];
    if (action != "load") { e.Response.StatusCode = 400; return; }
    var rows = _service.LoadPage(CurrentTenant, ParsePage(e.Request));
    e.Response.ContentType = "application/json";
    e.Response.Write(JsonSerializer.Serialize(rows));
}
```

Der Handler läuft im Kontext der Komponente und kann daher deren Zustand und die aktuelle Session lesen. Genau darin liegt sein ganzer Vorteil: Die Datenabfrage weiß, welcher Mandant, welche Filter und welche Berechtigungen gelten, ohne dass irgendetwas davon aus dem Browser übergeben werden muss.

WebMethod für Aufrufe im RPC-Stil

Ist der Aufrufer das eigene JavaScript der Integration und nicht die Datenquelle eines Anbieters, ist eine URL die falsche Form. Eine mit `[WebMethod]` markierte Methode lässt sich vom Client wie eine Funktion aufrufen: Typisierte Argumente werden aus JavaScript gemarshallt, ein Rückgabewert wird zurückgemarshallt, und der Client kann auf das Ergebnis warten. Sie kann statisch sein oder eine Instanzmethode an einem Container der obersten Ebene wie der Seite oder dem Fenster.

Das Lab implementiert denselben schreibgeschützten Datensatz auf beide Arten, um die Abwägung sichtbar zu machen. Die Postback-URL ist richtig, wenn die Anbieterbibliothek das Abrufen selbst übernimmt und eine Adresse erwartet, die sie mit ihren eigenen Paging-Argumenten aufrufen kann. Die WebMethod ist richtig, wenn Wrapper-Code einen Wert braucht – etwa um einen eingegebenen Code zu validieren oder eine Nachschlageliste abzurufen – und sonst eine rohe HTTP-Antwort parsen müsste.

Beide sind Einstiegspunkte in den Server, und beide umgehen die normale Event-Pipeline. Deshalb brauchen beide dieselbe Validierung und Autorisierung, die auch ein Button-Handler bekäme.

Parameter, Content-Types und Sicherheit

Jeder Datenendpunkt nimmt Eingaben aus dem Browser entgegen, und der Browser lässt sich von jedem skripten. Der Action-Parameter wird mit einer festen Liste verglichen und nicht verwendet, um daraus einen Methodennamen zu bauen. Paging-Argumente werden als Ganzzahlen mit einer Obergrenze geparkt, sodass eine Anfrage nach einer Million Zeilen abgewiesen wird. Filter- und Sortierwerte werden auf bekannte Spalten abgebildet, statt in eine Abfrage eingesetzt zu werden. Alles Unerwartete liefert einen Client-Fehler zurück und wird mit Session- und Korrelations-ID protokolliert.

Der Content-Type wird bei jeder Antwort ausdrücklich gesetzt, denn ein als Text ausgelieferter JSON-Body verwirrt manche Bibliotheken, und ein als JSON ausgelieferter HTML-Body verwirrt sie alle. Die Antwort auf die Frage aus dem Selbstcheck lautet `application/json` für JSON. Dateiantworten setzen den Disposition-Header und den tatsächlichen MIME-Typ, damit der Browser sie korrekt verarbeitet.

- Darf der aktuelle Benutzer diese Zeilen in diesem Mandanten sehen – geprüft im Handler?
- Sind Paging- und Filterparameter begrenzt und auf eine Whitelist abgebildet?
- Gibt eine Fehlerantwort über den Statuscode hinaus nichts über den Server preis?



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:

- ✓ ``getPostbackUrl`` / ``GetPostbackURL`` verwenden, um Browseranfragen an eine Komponenteninstanz zu leiten.
- ✓ Anfragen mit ``WebRequest`` verarbeiten und JSON, Dateien, Bilder oder jede andere vom Widget erwartete Antwort zurückgeben.

- ✓ `[WebMethod]`-Methoden für Client-Aufrufe bereitstellen, die sich wie RPC statt wie rohe HTTP-Anfragen verhalten sollen.
- ✓ Datenanfragen von Widgets mit Action-Parametern, Validierung, Content-Types und Sicherheitsprüfungen absichern.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Implementieren Sie für denselben schreibgeschützten Datensatz sowohl einen Postback-Endpunkt als auch einen WebMethod-Endpunkt. Verbinden Sie mit jedem ein einfaches Widget und vergleichen Sie die Vor- und Nachteile.

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

- ✓ Postback-Handler für WebRequest
- ✓ WebMethod mit Argumenten und Rückgabewert
- ✓ Vergleichsnotiz: URL-Datenquelle vs. Client-Callback

Selbstcheck vor der Abgabe

- ✓ Wann ist Postback besser als WebMethod?
- ✓ Warum sollten Action-Parameter validiert werden?
- ✓ Welchen Content-Type sollten JSON-Antworten verwenden?