

## Kapitel 31

# AJAX-Services

### In diesem Kapitel:

AJAX-Services mit ASMX-Webservices	680
AJAX-Services mit WCF	688
AJAX-Services mit Seitenmethoden	690
Typische Stolpersteine	691

An dem Beispiel der Fortschrittsanzeige im Kapitel »Partielle Seitenerzeugung mit dem UpdatePanel« wurde deutlich, dass auch bei der partiellen Seitenerzeugung der Overhead immer noch groß sein kann: Jedes Mal sendet der Server die im Panel enthaltenen HTML-Tags zur Erzeugung der Balken zum Browser. Eigentlich braucht der Browser aber nur eine einzige Zahl, aus der er die Darstellung selbst erzeugen könnte. ASP.NET AJAX bietet daher ein zweites Verfahren an, bei dem einzelne elementare Daten oder auch komplexere Datenstrukturen ohne die Layoutinformation im Stil eines Remote Procedure Call (RPC) zum Browser übermittelt werden. Auf der Serverseite definiert der Entwickler eine .NET-Methode mit Parametern und Rückgabewert. Diese Methode wird über eine spezielle Vorkennung als ein *AJAX-Service* auf dem Webserver bereitgestellt. Der Client (Webbrowser) erhält über einen generierten Proxy die Möglichkeit, diese Methode im AJAX-Service wie jede andere JavaScript-Methode aus JavaScript heraus aufzurufen. Die dem Methodenaufruf zugrunde liegende Verwendung des `XmlHttpRequest`-Objekts verbirgt der Proxy ebenso vor dem Benutzer wie die browserspezifischen Eigenarten dieses Objekts.

Es gibt drei Möglichkeiten zur Realisierung von AJAX-Fernaufrufen in ASP.NET 3.5:

- Webservices in Form von *.asmx*-Dateien (ASP.NET-basierte Webservices, alias Script Services)
- Webservices in Form von *.svc*-Dateien (WCF-basierte Webservices)
- Statische Methoden der Seitenklasse (Seitenmethoden – engl.: Page Methods)

---

**HINWEIS** Diese drei Möglichkeiten kann man in einer Seite miteinander mischen.

---

## AJAX-Services mit ASMX-Webservices

Das Interessante an diesen Webfernaufrufen ist, dass Microsoft auf einem Konzept aufsetzt, welches .NET ohnehin schon seit Version 1.0 bietet: ASP.NET-basierte Webservices (ASMX). Ein mit ASP.NET erzeugter XML-Webservice tauscht normalerweise SOAP-Nachrichten aus und stellt die Metadaten als WSDL-Dokument bereit. Für die AJAX Library hat Microsoft sich aber für ein Verfahren entschieden, das dem Browser die Last der Auswertung von XML-Dokumenten abnimmt. Wenn der Entwickler einen XML-Webservice zusätzlich mit der Annotation `[ScriptService]` ausstattet, erzeugt der Webservice durch Anhängen von */js* an den URL plötzlich JavaScript-Code für einen Browser statt WSDL und serialisiert statt im SOAP-Format in Gestalt der JavaScript Object Notation (JSON). Diesen Ablauf veranschaulicht Abbildung 31.1.

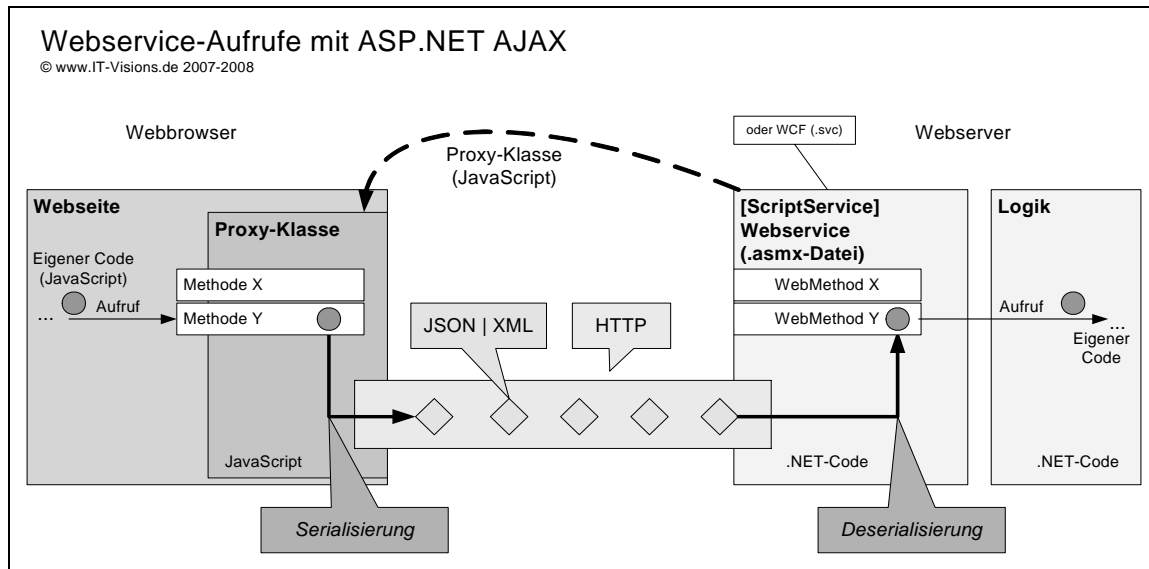


Abbildung 31.1 AJAX-Aufrufe von ASMX-Webservices

Das auch in diesem Fall notwendige ScriptManager-Steuerelement erhält in der Sektion `<ServiceReferences>` den URL des Webservice. Visual Studio lässt die Wahl, ob der von dem Webservice erzeugte JavaScript-Proxy-Code jeweils ad hoc von dem Webservice abgerufen oder in die Webseite eingebettet werden soll. Auf den Webservice kann man dann aus einer beliebigen JavaScript-Routine über den Namen der Klasse und der Methode zugreifen.

Dabei hat die JavaScript-Methode aber immer zwei Parameter mehr als die ursprüngliche Methode. Die zusätzlichen beiden Parameter erfüllen das *A* in AJAX, die *Asynchronität*. Der Aufrufer erhält keinen Rückgabewert von dem Methodenaufruf, sondern gibt zwei JavaScript-Routinen an, die im Erfolgs- bzw. Fehlerfall aufzurufen sind.

**ACHTUNG** WCF-Dienste können in der ASP.NET AJAX-Version 1.0 noch nicht verwendet werden. Dies will Microsoft aber in der nächsten Version von ASP.NET AJAX ergänzen. Zu beachten ist ferner, dass man immer nur Webservices auf dem gleichen Webserver aufrufen kann, auf dem auch die Webseite liegt. Dies ist eine Sicherheitsfunktion des `XmHttpRequest`-Objekts. Zum Aufruf von Webservices auf anderen Webservern (z.B. für Mashups) muss der Webentwickler einen lokalen Wrapper erstellen. In einer zukünftigen ASP.NET-Version will Microsoft dies durch deklarierbare Webservicebridges vereinfachen.

## Beispiele

In dem folgenden Beispiel werden drei AJAX-Webserviceaufrufe verwendet:

- **Flugnummernsuche:** Beim Klick auf die *Prüfen*-Schaltfläche werden Informationen zu dem in das Textfeld eingegebenen Flug abgeholt.
- **Abflugortauswahl:** Bei der Auswahl eines Abflugortes wird die Liste der erreichbaren Zielorte geholt.
- **Zielauswahl:** Bei der Auswahl eines Zielortes wird der nächste verfügbare Flug angezeigt.

**Abbildung 31.2** Beispiel, in dem ASP.NET AJAX-Webserviceaufrufe verwendet werden

Das umfangreiche Listing zu der oben skizzierten Aufgabe wird hier in Ausschnitten und verschiedenen Teilbereichen kommentiert wiedergegeben.

## ASMX-Webservice

Das folgende Listing zeigt einen mit `[ScriptMethod]` annotierten ASMX-Webservice, der im Hintergrund der Lösung arbeitet. Durch nur eine einzige zusätzliche Codezeile wird aus einem ASMX-Webservice ein Endpunkt für AJAX-Aufrufe des Webbrowsers:

```
[WebService(Namespace="http://IT-Visions.de/wwwings", Name="WorldWideWings Flugplan-Webservice",
Description="Webservices für den WorldWideWings-Flugplan!"),
WebServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1, EmitConformanceClaims=true)]
[System.Web.Script.Services.ScriptService()]
public class FlugplanService : System.Web.Services.WebService
{
    [WebMethod()]
    public de.WWWings.Flug GetFlight(long FlightNo)
    {
        return de.WWWings.FlugBLManager.HoleFlug(FlightNo);
    }
    [WebMethod()]
    public string[] GetDestinations(string Abflugort)
    {
        return de.WWWings.FlugBLManager.Ziele_Fuer_Abflugort(Abflugort);
    }
}
```

**Listing 31.1** Einsatz von `[ScriptService]`

**HINWEIS** Vergessen Sie nicht die Annotation `[ScriptService]`. Sonst bekommen Sie den Laufzeitfehler: »Only web services with a `[ScriptService]` attribute on the class definition can be called from script.«

## ScriptManager

Das ScriptManager-Steuerelement in der Seite erhält einen Verweis auf den Webservice. Durch `InlineScript=true` wird festgelegt, dass der Proxy in den Quelltext der Seite eingebunden werden soll:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference InlineScript="True" Path="/Webservice/FlugplanService_ASMXAJAX.aspx">
    </asp:ServiceReference>
  </Services>
</asp:ScriptManager>
```

Listing 31.2 Verweis auf einen ASMX-Webservice im *ScriptManager*-Steuerelement

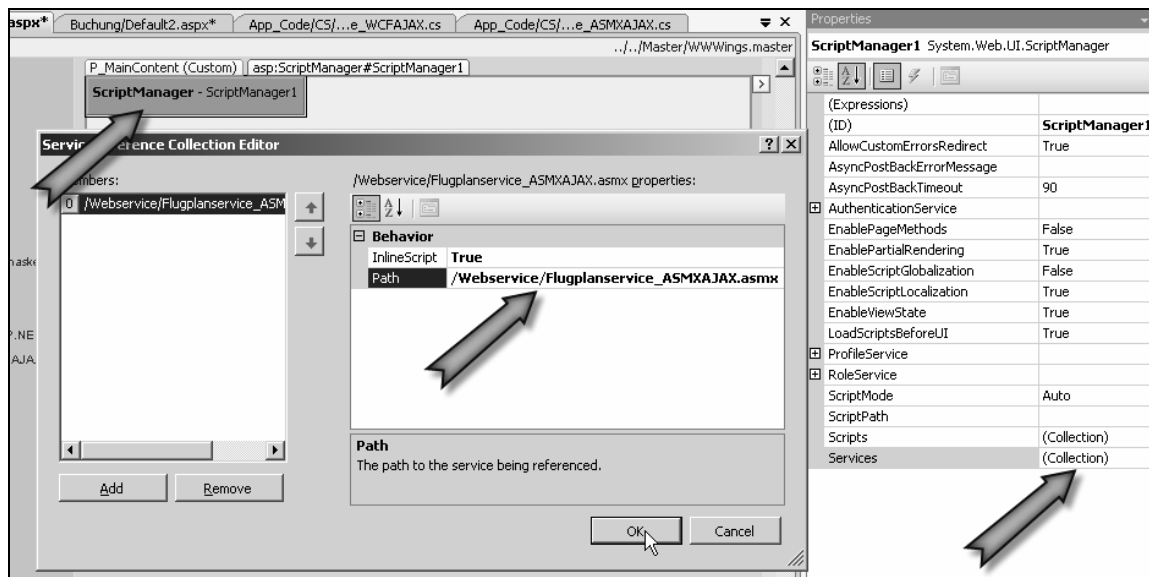
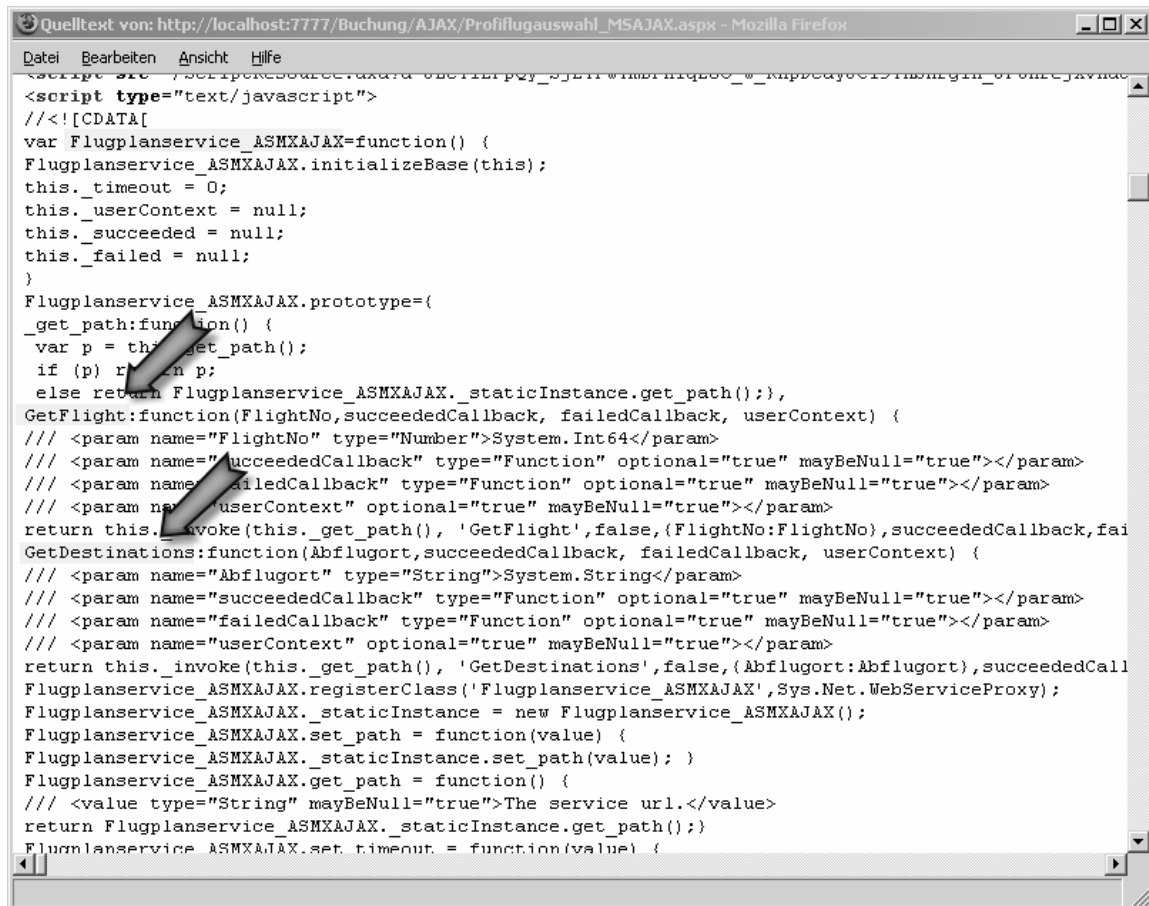


Abbildung 31.3 Festlegung einer *ScriptReference* in Visual Studio 2008

Abbildung 31.4 zeigt Ausschnitte des erzeugten Proxys. Die JavaScript-Repräsentation der beiden Webserviceoperationen sind in der Abbildung durch Pfeile markiert.



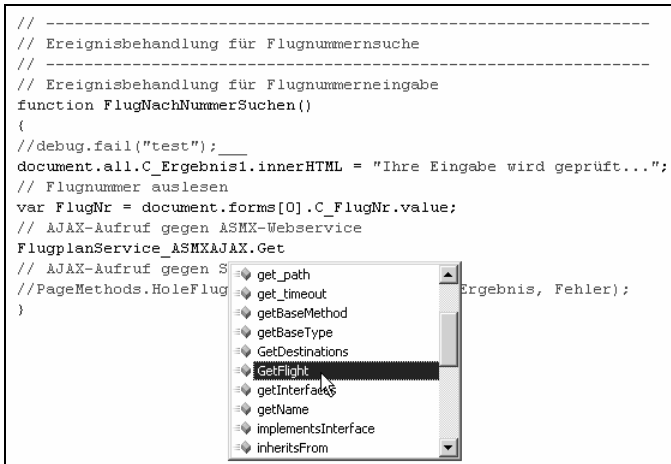


Abbildung 31.5 IntelliSense für Webserviceproxys

**TIPP**

Leider funktioniert IntelliSense für die Webservices nicht immer auf Anhieb. Wenn sie nicht arbeitet, können Sie Folgendes versuchen:

- Wechseln in die Designansicht und zurück zur Codeansicht
- Schließen der Datei und wieder öffnen
- Komplette Kompilierung der Website
- Schließen des Projekts in Visual Studio und erneut öffnen
- Schließen von Visual Studio und erneut öffnen
- Und vor allem: Geduld haben!

Bei einem Aufruf einer Webserviceoperation ist zu beachten, dass man mindestens einen Parameter mehr angeben muss, als die Operation eigentlich benötigt. Der Grund dafür ist, dass AJAX asynchron ist. Der zusätzliche Parameter ist der Name einer JavaScript-Funktion, die beim Eingang des Ergebnisses vom Server aufgerufen werden soll. Man kann optional aber auch zwei oder drei Parameter angeben. Der zweite zusätzliche Parameter ist der Name einer Funktion, die im Fehlerfall aufgerufen werden soll. Der dritte Name ist eine Zeichenkette (genannt Kontext), die dem Ergebnis übergeben werden soll. Dies ist hilfreich, wenn mehrere verschiedene Aufrufe von einer Rückruffunktion behandelt werden sollen.

Die Rückruffunktion hat einen, zwei oder drei Parameter:

- Der erste Parameter ist der Rückgabewert der serverseitigen Methode.
- Der zweite (optionale) Parameter ist die beim Aufruf mit übergebene Zeichenkette (Kontext).
- Der dritte (optionale) Parameter ist der Name der aufgerufenen Methode.

In dem folgenden Listing sieht man anschaulich, dass das von dem JavaScript-Proxy gelieferte Ergebnis des AJAX-Aufrufs keine Sammlung von Einzelwerten, sondern ein JavaScript-Objekt mit der gleichen Struktur wie das serverseitige .NET-Objekt Flug ist, also die Attribute FlugNr, AbflugOrt, Zielort etc. besitzt:

```
// -----
// Hilfsroutinen
// -----
```

```

// Wird aufgerufen, wenn es auf dem Server zu einem Fehler kam
function Fehler( error )
{
    alert("Fehler bei der AJAX-Verarbeitung auf dem Server: " + error.get_message());
}

// -----
// Ereignisbehandlung für Flugnummernsuche
// -----
// Ereignisbehandlung für Flugnummerneingabe
function FlugNachNummerSuchen()
{
    //debug.fail("test");
    document.all.C_Ergebnis1.innerHTML = "Ihre Eingabe wird geprüft...";
    // Flugnummer auslesen
    var FlugNr = document.forms[0].C_FlugNr.value;
    // AJAX-Aufruf gegen ASMX-Webservice
    FlugplanService_ASMXAJAX.GetFlight(FlugNr,FlugNachNummerSuchen_Ergebnis,Fehler);
    // AJAX-Aufruf gegen Seitenmethode
    //PageMethods.HoleFlug(n,FlugNachNummerSuchen_Ergebnis, Fehler);
}

// -----
// Ereignisbehandlung bei Eingang des Ergebnisses vom Server für Flugnummernsuche
// -----
function FlugNachNummerSuchen_Ergebnis(result)
{
    document.all.C_Link.href="ajax/Fortschrittsanzeige_ATLAS.aspx?id=" + result.FlugNr;
    ausgabe = "Sie haben gewählt:<br>Flug " + result.FlugNr + " von " + result.AbflugOrt + " nach " +
    result.ZielOrt + " und hat " + result.FreiePlaetze + " freie Plätze!";
    document.all.C_Ergebnis1.innerHTML = ausgabe;
    document.all.C_Link.style.visibility = "visible"
}

// -----
// Ereignisbehandlung für Abflugauswahl
// -----
function AbflugortAuswahl() {
    C_Abflugort = $get('<%= this.C_Abflug.ClientID %>');
    // Ermittlung der Option
    var abflug = C_Abflugort.options[C_Abflugort.selectedIndex].value;
    // Callback zum Server
    document.all.C_Ergebnis2.innerHTML = "Lade Zielorte...";
    FlugplanService_ASMXAJAX.GetDestinations(abflug,AbflugortAuswahl_Ergebnis);
}

// -----
// Ereignisbehandlung bei Eingang des Ergebnisses vom Server für Abflugort
// -----
function AbflugortAuswahl_Ergebnis(result)
{
    C_Zielort = document.getElementById('<%= this.C_Zielort.ClientID %>');
    document.all.C_Ergebnis2.innerHTML = result.length + " Zielorte wurden gefunden.";
    // Leeren der Liste
    while (C_Zielort.options.length > 0) { C_Zielort.remove(0); }
    // Füllen der Liste
    for (var i=0; i<result.length; i++) {

        //C_Zielort.visible = true;
        var Option = document.createElement("option");
        Option.value = result[i];
        Option.text = result[i];
    }
}

```



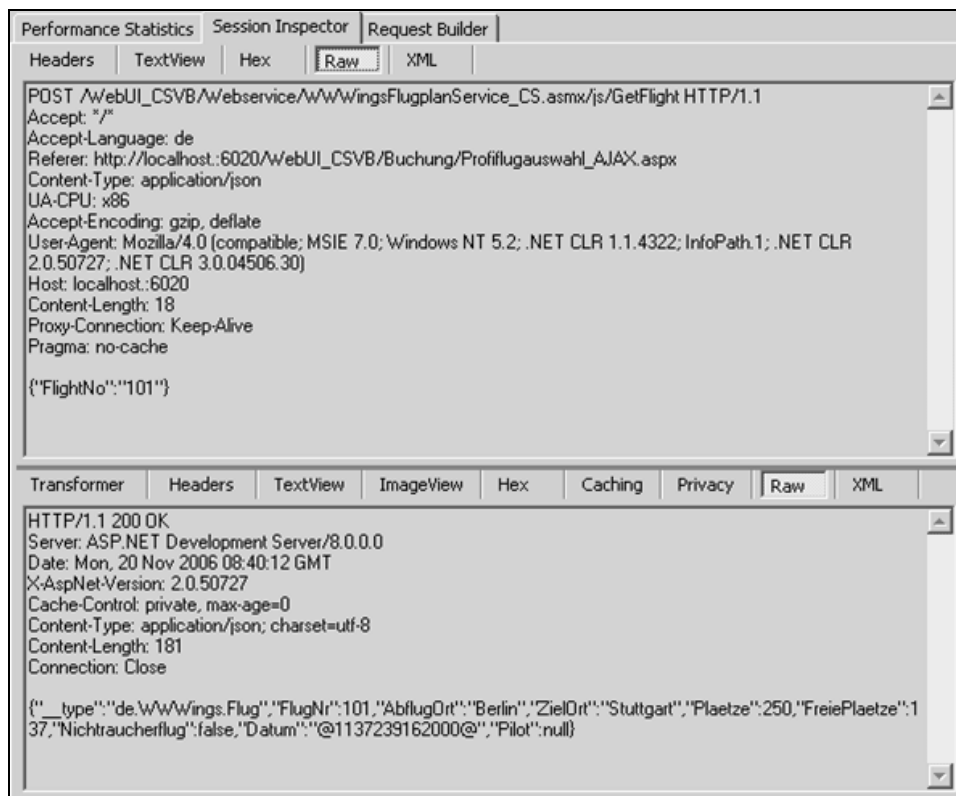
```

C_Zielort.add(Option);
}
// -----
// Ereignisbehandlung für Zielauswahl
// -----
// Ereignisbehandlung für Zielortauswahl
function ZielortAuswahl() {
C_Abflugort = document.getElementById('<%= this.C_Abflug.ClientID %>');
C_Zielort = document.getElementById('<%= this.C_Zielort.ClientID %>');
// Ermittlung der Option
var abflugort = C_Abflugort.options[C_Abflugort.selectedIndex].value;
var zielort = C_Zielort.options[C_Zielort.selectedIndex].value;
ausgabe = "Sie haben gewählt:<br>Route von " + abflugort + " nach " + zielort;
document.all.C_Ergebnis1.innerHTML = ausgabe;
document.all.C_Link.href="ajax/Fortschrittsanzeige_ATLAS.aspx?von=" + abflugort + "&nach=" + zielort;
document.all.C_Link.style.visibility = "visible"
}
</script>

```

**Listing 31.3** Das clientseitige Skript für das Beispiel (JavaScript)

Bei dem AJAX-Aufruf eines ASMX-Webservice sind die HTTP-Header größer als die eigentliche Nutzlast, wie man im Netzwerkmonitor Fiddler [FID01] (Abbildung 31.6) erkennt.



**Abbildung 31.6** Ein AJAX-Webserviceaufruf mit einem HTTP-Monitor betrachtet

**TIPP** Neben der Möglichkeit zum Aufruf von Webservices, die in eigenständigen ASMX-Dateien realisiert sind, kann der AJAX-Entwickler auch die Option nutzen, Methoden in einer normalen ASPX-Seite via AJAX aufzurufen: Dazu muss die Methode mit `[WebMethod]` annotiert sein (im Unterschied zu `[ScriptService]` bei `.asmx`-Dateien) und dann im Browser über den feststehenden Objektnamen `PageMethods` angesprochen werden.

ASP.NET stellt AJAX als vordefinierte Webservices die Authentifizierung und den Personalisierungsdienst von ASP.NET zur Verfügung. Somit kann der Webentwickler den Anmeldedialog und die Personalisierung von Webseiten durch AJAX realisieren.

## Timeout

Die maximale Wartezeit des Browsers auf das Ergebnis eines Webserviceaufrufs kann man vor dem Aufrufen der Methode im Proxy festlegen:

```
FlugplanService_ASMXAJAX.set_timeout(1500)
```

bzw.

```
PageMethods.set_timeout(1500);
```

Die Angabe erfolgt in Millisekunden. Wenn die Dauer überschritten ist, wird die Fehlerrückrufmethode aufgerufen.

## AJAX-Services mit WCF

Seit ASP.NET 3.5 kann man alternativ zu ASMX-Webservices auch WCF-Webservices angeben. Microsoft hat in .NET 3.5 für die WCF ein neues Binding mit Namen `webHttpBinding` eingeführt, das HTTP als Transportprotokoll und JSON als Serialisierungsformat unterstützt.

Für diese Variante legt man einen neuen WCF-Dienst mit der Elementvorlage *AJAX-enabled WCF-Service* an. Dabei entsteht eine `.svc`-Datei mit einer zugehörigen Hintergrundcodedatei. Außerdem wird in die Konfigurationsdatei im Wurzelverzeichnis eine Sektion `<system.servicemodel>` eingefügt, die das Binding beschreibt. Diese Sektion kann auch in die `web.config`-Datei eines Unterverzeichnisses verschoben werden.

**TIPP** Auch hier kann man den generierten JavaScript-Proxy mit Anhängen von `/js` oder `/jsdebug` an den URL betrachten.

```
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name="FlugplanServiceAJAXAspNetAjaxBehavior">
        <enableWebScript />
      </behavior>
    </endpointBehaviors>
  </behaviors>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
  <services>
    <!-- AJAX Dienste -->
    <service name="FlugplanService_WCFAJAX">
```

```
<endpoint address="" behaviorConfiguration="FlugplanServiceAJAXAspNetAjaxBehavior"
  binding="webHttpBinding" contract="FlugplanService_WCFAJAX" />
</service>
</services>
</system.ServiceModel>
```

**Listing 31.4** Konfigurationseinträge für einen AJAX-fähigen WCF-Dienst

```
using System;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
public class Flugplanservice_WCFAJAX
{
    [OperationContract]
    public void DoWork()
    {
        return;
    }

    [OperationContract]
    public de.WWWings.Flug GetFlight(long FlightNo)
    {
        return de.WWWings.FlugBLManager.HoleFlug(FlightNo);
    }

    [OperationContract]
    public string[] GetDestinations(string Abflugort)
    {
        return de.WWWings.FlugBLManager.Ziele_Fuer_Abflugort(Abflugort);
    }
}
```

**Listing 31.5** Implementierung eines WCF-Dienstes

Die *.svc*-Datei ist anschließend genau wie eine *.asmx*-Datei in den ScriptManager als *<ServiceReference>* einzubinden.

---

**ACHTUNG** Bei der Einbindung von WCF-Diensten ist aber zu beachten, dass *InlineScript* in *<ServiceReference>* nicht auf *true* gesetzt werden darf.

---

```
<asp:ScriptManager EnablePageMethods="true" ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference InlineScript="false"
      Path="/Webservice/Flugplanservice_WCFAJAX.svc">
    </asp:ServiceReference>
  </Services>
</asp:ScriptManager>
```

**Listing 31.6** Verweis auf einen WCF-Webservice im *ScriptManager*-Steuerelement

Der Aufruf des WCF-Webservices erfolgt dann analog zum ASMX-Webservice. Bei komplexen Datentypen muss man jedoch einen wesentlichen Unterschied beachten: Der XML-Serialisierer, der in ASMX zum Einsatz kommt, serialisiert die öffentlichen Properties einer Klasse. Der *NetDataContract-Serialisierer* von WCF serialisiert aber die Fields einer Klasse. Daher muss man in dem obigen Beispiel bei der Ausgabe der Flüge nun statt

```
// ASMX und PageMethod:
ausgabe = "Sie haben gewählt:<br>Flug " + result.FlugNr + " von " + result.AbflugOrt + " nach " +
result.ZielOrt + " und hat " + result.FreiePlaetze + " freie Plätze!";
```

jetzt schreiben:

```
// WCF:
ausgabe = "Sie haben gewählt:<br>Flug " + result.flugNr + " von " + result.abflugOrt + " nach " +
result.zielOrt + " und hat " + result.freiePlaetze + " freie Plätze!";
```

## AJAX-Services mit Seitenmethoden

Der Aufruf eines ASMX-Webservice oder WCF-Webservice in einer AJAX-Seite ist eine gute Möglichkeit, Methoden zwischen verschiedenen Seiten wieder verwendbar zu machen. Wenn man aber eine Methode nur genau einer Seite zuordnen möchte, dann ist das Anlegen eines separaten Webservice nicht nur lästig, sondern macht die Lösung auch unübersichtlicher.

Seitenmethoden (engl. *Page Methods*) sind Methoden in einer ASPX-Seite, die zum Aufruf per AJAX zugelassen sind. Diese Methoden müssen mit der Annotation `[WebMethod]` versehen werden. Dabei können die Methoden in der Code-Behind-Datei oder auch in einem serverseitigen Skriptblock in der *.aspx*-Datei liegen und die gleichen (komplexen) Datentypen zurückgeben wie ASMX-Webservices. Die Serialisierung der Typen erfolgt wie bei ASMX-Webservices.

```
[System.Web.Services.WebMethod]
public static de.WWWings.Flug GetFlight(long FlightNo)
{
    return de.WWWings.FlugBLManager.HoleFlug(FlightNo);
}
```

**Listing 31.7** Erstellen einer Seitenmethode

Bevor eine Seitenmethode angesprochen werden kann, muss man im *ScriptManager*-Steuerelement diese Funktion grundsätzlich über *EnablePageMethods* aktivieren:

```
<asp:ScriptManager EnablePageMethods="true" ID="ScriptManager1" runat="server">
```

Die Seitenmethode kann danach im JavaScript-Code über den feststehenden Objektnamen PageMethods angesprochen werden:

```
PageMethods.GetFlight(FlugNr, FlugNachnummerSuchen_Ergebnis, Fehler);
```

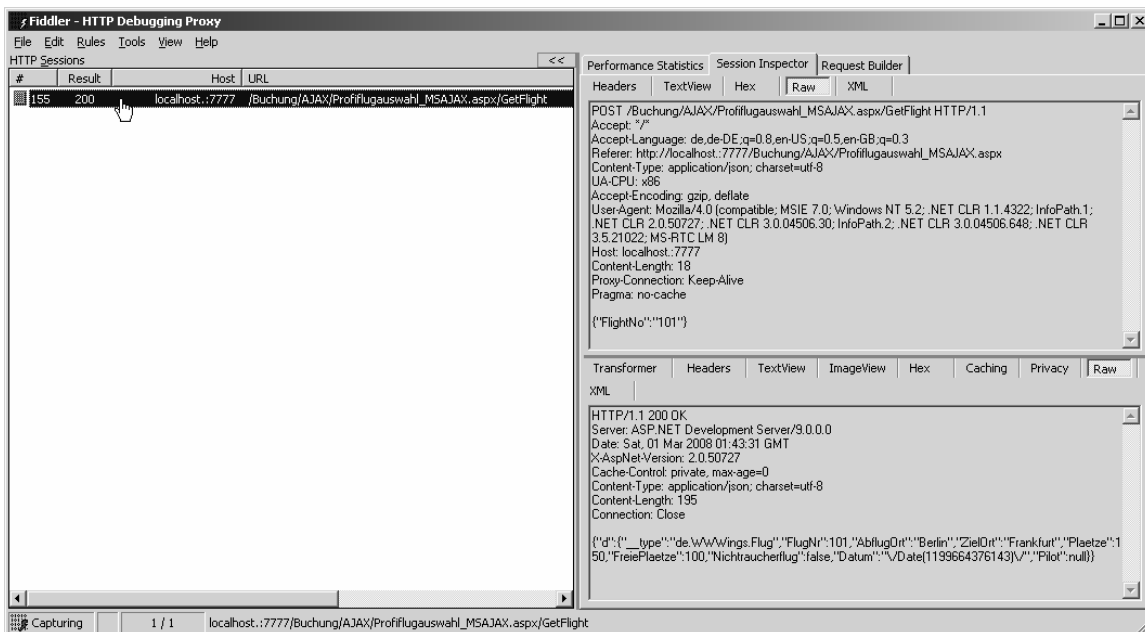


Abbildung 31.7 Datenübertragung beim Aufruf der Seitenmethode GetFlight()

## Typische Stolpersteine

Hier finden Sie eine Liste der typischen Schwierigkeiten, die bei ASP.NET AJAX auftreten können, und Hinweise, wie Sie diese vermeiden:

- Bei JavaScript wird zwischen Groß- und Kleinschreibung unterschieden. Die Groß- und Kleinschreibung der serverseitigen Bezeichner für Klassen, Methoden und Attribute bleibt bei der Generierung der JavaScript-Proxys erhalten und muss also im Clientskript genauso aussehen wie auf dem Server.
- Bei den AJAX-Services auf Basis von ASMX-Webservices vergisst man oft die Annotation [ScriptService]. Der direkte Test des Webservice im Browser funktioniert dann zwar, es ist aber kein JavaScript-Aufruf möglich. Zum Test, ob [ScriptService] aktiviert ist, rufen Sie den Dienst auf mit dem Zusatz /JS, z. B. *http://server/dienst.aspx/JS*. Dann sollten Sie eine JavaScript-Datei mit dem Proxy erhalten.
- Man kann immer nur Dienste auf dem Webserver aufrufen, von dem auch die Webseite geladen wurde, in der das JavaScript läuft. Dies ist eine Sicherheitsvorkehrung aller Browser (Same Origin Policy).

- Die maximale Übertragungsgröße ist standardmäßig begrenzt. Bei AJAX-Services auf Basis von ASMX oder Seitenmethoden sind dies 2.097.152 Byte. Die Größe kann in der Konfigurationsdatei verändert werden:

```
<system.web.extensions>
<scripting>
<webServices>
  <jsonSerialization maxJsonLength="500">
    <converters>
      <add name="ConvertMe" type="Acme.SubAcme.ConvertMeTypeConverter"/>
    </converters>
  </jsonSerialization>
</webServices>
</scripting>
</system.web.extensions>
```

**Listing 31.8** Einstellen der maximalen Übertragungsgröße für JSON