ASP.NET AJAX. Programowanie w nurcie Web 2.0

Autor: Christian Wenz
Tłumaczenie: Marek Pałczyński
ISBN: 978-83-246-1494-3
Tytuł oryginału: Programming ASP.NET AJAX: Build rich, Web 2.0-style UI with ASP.NET AJAX
Format: 168x237, stron: 432

Wykorzystaj najlepsze rozwiązania technologii AJAX i stwórz interaktywną stronę internetową

- Jak wykorzystywać dane serwerowe?
- Jak tworzyć i udostępniać własne kontrolki?
- Jak aktualizować część strony w regularnych odstępach czasu?

Zastanawiałeś się, dlaczego interaktywne witryny cieszą się dziś taką popularnością? Dzieje się tak głównie dlatego, że wymagają one od użytkowników współuczestnictwa w tworzeniu i rozwoju serwisu, a tym samym powodują, że abonenci mają duży wpływ na jego ostateczny kształt. Dzięki temu każdy odbiorca korzysta z atrakcyjnej witryny idealnie dopasowanej do swoich potrzeb. To właśnie ASP.NET AJAX umożliwia projektowanie profesjonalnych, interaktywnych stron WWW w duchu Web 2.0. Znawcy tematu zapewniają, że AJAX jest rozwiązaniem przyszłościowym w dziedzinie projektowania serwisów internetowych. O tym, jak za pomocą tej technologii wdrożyć w swoim serwisie rozwiązania zgodne z filozofią Web 2.0, dowiesz się właśnie z tego podręcznika.

W książce „ASP.NET AJAX. Programowanie w nurcie Web 2.0" zamieszczono, oprócz teoretycznych wiadomości, mnóstwo przykładów demonstrujących działanie najważniejszych mechanizmów środowiska ASP.NET AJAX. Przedstawione rozwiązania mają bardzo ogólny charakter, a zatem możesz szybko dostosować je do potrzeb własnej aplikacji. Korzystając z tego podręcznika, nauczysz się m.in. projektować własne kontrolki i udostępniać je w serwisie Toolkit, poznasz zasady korzystania ze standardowych bibliotek AJAX-a w innych środowiskach (np. PHP). Będziesz umiał zbudować profesjonalną, dynamiczną stronę internetową, bazującą na platformie ASP.NET AJAX.

- Struktura i architektura środowiska ASP.NET AJAX
- JavaScript
- Rozszerzenia ASP.NET AJAX
- Usługi sieciowe
- Odświeżanie części strony — obiekt UpdatePanel
- Lokalizacja i globalizacja aplikacji
- ASP.NET Control Toolkit
- Animacja na stronie WWW
- Wiązanie i walidacja danych
- Zachowania i komponenty
- Dokumentacja klasy XMLHttpRequest i modelu DOM

Płyń z nurtem nowoczesności — twórz elektryzujące, interaktywne strony WWW!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 5.

Usługi sieciowe

Usługa sieciowa została wykorzystana już w pierwszym rozdziale książki w przykładzie aplikacji „Witaj świecie". Jej zadanie polegało wówczas na przekazywaniu danych między klientem i serwerem. Chcąc jednak skorzystać ze wszystkich możliwości, jakie daje połączenie usług sieciowych ze skryptami JavaScript, trzeba się zapoznać z kilkoma bardziej zaawansowanymi sposobami wykorzystywania tego typu rozwiązań. Zaliczają się do nich między innymi: obsługa błędów, stosowanie osadzanych usług sieciowych (metod usług sieciowych zawartych w kodzie strony *.aspx*, zwanych też czasami metodami strony) oraz wykorzystanie usług sieciowych i skryptów JavaScript bez wsparcia ze strony platformy .NET.

W tym rozdziale zostaną przedstawione pewne szczególne rozwiązania środowiska ASP.NET AJAX związane z obsługą usług sieciowych, w tym procedury obsługi błędów oraz przechowywanie informacji o stanie sesji. Tematyka rozdziału obejmuje również zasady odwoływania się z poziomu skryptów JavaScript do usług sieciowych, które nie zostały przygotowane w środowisku ASP.NET.

Obsługa błędów

W analizowanych wcześniej przykładach zakładaliśmy, że wywołania zdalnych metod zawsze kończą się poprawnie. Nie uwzględnialiśmy możliwości wygenerowania wyjątku.

Projektanci serwisów internetowych często pomijają procedury obsługi błędów w przypadku odwołań do usług sieciowych udostępnianych przez zdalne serwery (czyli serwery pracujące w innej domenie). Jedną z przyczyn jest to, że usługi sieciowe można implementować na podstawie różnych technologii, a każda z technologii dysponuje własnym mechanizmem zgłaszania wyjątków, a niektóre z nich w ogóle nie generują wyjątków.

W przypadku platformy ASP.NET AJAX i rozwiązań Ajax praca z usługami sieciowymi odbiega nieco od standardowego modelu. Nie można wywoływać bezpośrednio usługi sieciowej, ponieważ zabrania tego system bezpieczeństwa. Domyślnie interpreter JavaScript i obiekt `XMLHttpRequest` pozwalają jedynie na odwołania z użyciem adresów URI z tej samej domeny, z której pochodzi strona. Zatem podczas pracy w środowisku ASP.NET AJAX wywołania usług sieciowych są kierowane do serwera w tej samej domenie. To z kolei oznacza, że sama usługa sieciowa opiera się na technologii .NET (lub WCF — nowym modelu Windows Communication Foundation). Zasady generowania wyjątków są więc znane.

Zapewnienie dostępu do wyjątków generowanych przez usługi sieciowe w skryptach JavaScript należy do zadań platformy ASP.NET AJAX. Aby sprawdzić działanie opisywanego mechanizmu, możemy utworzyć usługę matematyczną, która będzie dzieliła dwie liczby. Doprowadzenie do wygenerowania wyjątku nie będzie trudne — wystarczy wymusić dzielenie przez zero, co powinno spowodować wywołanie przez usługę wyjątku `DivideByZeroException`. Kod usług sieciowej (*MathService.asmx*) został przedstawiony w przykładzie 5.1. Analizując treść przykładu, warto zwrócić uwagę na atrybuty `[ScriptService]` i `[WebMethod]`, które muszą być uwzględnione w każdej usłudze sieciowej ASP.NET AJAX.

Przykład 5.1. Usługa sieciowa generująca wyjątek

```
MathService.asmx

<%@ WebService Language="C#" Class="MathService" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://hauser-wenz.de/AspNetAJAX/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class MathService : System.Web.Services.WebService {
    [WebMethod]
    public float DivideNumbers(int a, int b) {
      if (b == 0) {
        throw new DivideByZeroException( );
      } else {
        return (float)a / b;
      }
    }
}
```

Przygotujmy stronę, która wywoła usługę sieciową. Potrzebne będą dwa pola edycyjne, w których użytkownik będzie wpisywał liczby do podzielenia oraz dwa obszary na dane wyjściowe — jeden na wynik działania matematycznego, a drugi na ewentualne komunikaty o błędach. Kod musi również obejmować przycisk wywołujący funkcję JavaScript, która następnie wywoła usługę sieciową.

```
<nobr>
  <input type="text" id="a" name="a" size="2" />
  /
  <input type="text" id="b" name="b" size="2" />
  =
  <span id="c" style="width: 50px;" />
</nobr>
<br />
<input type="button" value="Podziel liczby" onclick="callService(this.form);" />
<br />
<div id="output" style="width: 600px; height: 300px;">
</div>
```

Spośród kontrolek serwerowych na stronie trzeba umieścić komponent `ScriptManager` wraz z osadzonym w jego treści odniesieniem do wykorzystywanej usługi sieciowej.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="MathService.asmx" />
  </Services>
</asp:ScriptManager>
```

Dzięki takiemu rozwiązaniu wywołania usługi sieciowej mogą być realizowane za pomocą obiektu pośredniczącego o nazwie `MathService`, który zostanie wygenerowany automatycznie. Podczas wywoływania metody sieciowej konieczne jest zachowanie odpowiedniej kolejności parametrów. Najpierw definiowane są parametry (lub parametr) przekazywane do metody sieciowej, a następnie funkcja zwrotna wykonywana po zakończeniu wywołania metody.

Jednak tym razem do metody `DivideNumbers()` zostanie przekazany jeszcze jeden dodatkowy parametr. Za funkcją zwrotną, wykonywaną po zakończeniu wywołania, zostanie zdefiniowana jeszcze jedna funkcja zwrotna. Druga z funkcji zwrotnych będzie wywoływana w przypadku wystąpienia błędów (w tym również w przypadku upłynięcia dopuszczalnego czasu realizacji zadania).

```
function callService(f) {
  document.getElementById("c").innerHTML = "";
  MathService.DivideNumbers(
    parseInt(f.elements["a"].value),
    parseInt(f.elements["b"].value),
    callComplete,
    callError
  );
}
```

Funkcja obsługi błędów otrzymuje obiekt błędu zawierający pięć metod:

`get_exceptionType( )`
: Metoda ta udostępnia informacje o typie wyjątku.

`get_message( )`
: Metoda ta zwraca komunikat o błędzie związany z wyjątkiem.

`get_stackTrace( )`
: Metoda te zwraca informacje o stosie wywołań funkcji.

`get_statusCode( )`
: Metoda ta udostępnia kod statusowy przekazany przez serwer.

`get_timeOut( )`
: Metoda ta pozwala na ustalenie, czy został przekroczony maksymalny czas realizacji zadania.

Informacje na temat błędu są wyświetlane w obszarze elementu `<div>`, który został utworzony specjalnie w tym celu.

```
function callError(result) {
  document.getElementById("output").innerHTML =
    "<b>" +
    result.get_exceptionType( ) +
    "</b>: " +
    result.get_message( ) +
    "<br />" +
    result.get_stackTrace( );
}
```

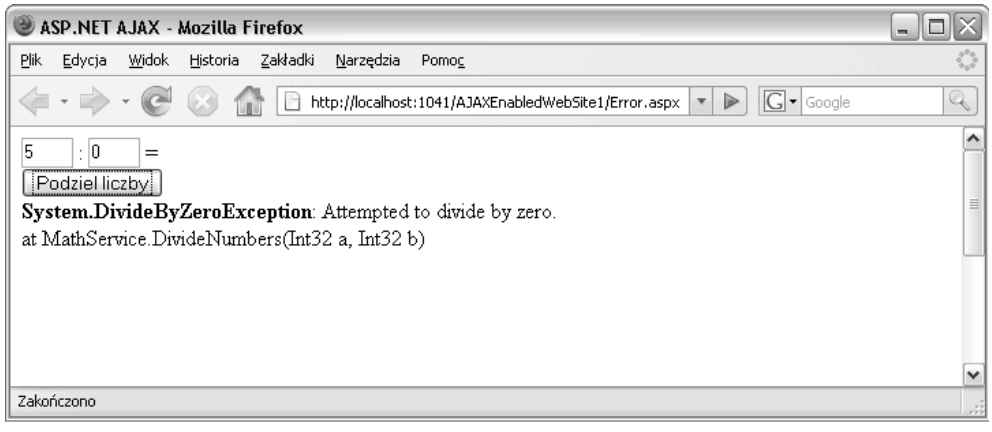
Przygotowanie pozostałej części kodu nie powinno nastręczać większych trudności. Gdy wywołanie usługi sieciowej zakończy się pomyślnie, wynik powinien zostać wyświetlony w obszarze elementu `<span>`. Pełna treść strony została przedstawiona w przykładzie 5.2.

Przykład 5.2. Strona wyświetlająca wyjątek wygenerowany przez usługę MathService.asmx

Error.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    document.getElementById("output").innerHTML = "";
    MathService.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result;
  }

  function callError(result) {
    document.getElementById("output").innerHTML =
      "<b>" +
      result.get_exceptionType() +
      "</b>: " +
      result.get_message() +
      "<br />" +
      result.get_stackTrace();
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="MathService.asmx" />
      </Services>
    </asp:ScriptManager>
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        =
        <span id="c" style="width: 50px;"></span>
      </nobr>
      <br />
      <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
      <br />
      <div id="output" style="width: 600px; height: 300px;">
      </div>
    </div>
  </form>
</body>
</html>
```

Podzielenie liczby 5 przez 6 daje spodziewany wynik 0.8333333. Jednak próba podzielenia liczby 5 przez 0 powoduje wygenerowanie przez usługę sieciową wyjątku, a w konsekwencji wyświetlenie komunikatu o błędzie wraz ze stosem wywołań funkcji (wygląd strony został pokazany na rysunku 5.1).



Rysunek 5.1. Wyświetlenie informacji na temat wyjątku

Informacja na temat (nie)wyświetlania komunikatów o błędach

Wyświetlanie komunikatów o błędach w aplikacji klienckiej jest doskonałym rozwiązaniem na czas uruchamiania aplikacji. Stanowi jednak bardzo duże zagrożenie w środowisku użytkowym. Komunikaty o błędach mogą bowiem zawierać tajne dane, takie jak parametry ciągów połączenia. Nawet jeśli nie są bezpośrednio wyświetlane w oknie przeglądarki środowisko ASP.NET AJAX może je przekazywać do aplikacji klienckiej. Aby temu zapobiec, należy wykonać dwie czynności. Po pierwsze trzeba sprawdzić, czy do przeglądarki nie są dostarczane szczegółowe opisy błędu (obejmujące dane na temat stosu wywołań funkcji). Po drugie generując wyjątek po stronie serwera należy uwzględnić w komunikacie możliwie najmniejszą ilość szczegółowych informacji.

Metody strony

Prawdopodobnie większość programistów zgodzi się z twierdzeniem, że umieszczanie wszystkich metod sieciowych aplikacji w oddzielnym pliku jest dość uciążliwe. Pod względem struktury aplikacji taki sposób zarządzania plikami wydaje się właściwy. Jednak w przypadku nieskomplikowanych skryptów i aplikacji (takich jak większość opisywanych w książce) dodatkowy plik *.asmx* niepotrzebnie rozbudowuje projekt.

Przy niewiele większym narzucie kodowym (lub nawet zmniejszeniu ilości kodu w pewnych okolicznościach) istnieje możliwość zamieszczenia całego skryptu w jednym miejscu — w głównym pliku *.aspx* (lub w związanym z nim pliku klasy). Procedura przygotowania opisywanego rozwiązania składa się z dwóch etapów. Pierwszy sprowadza się do zaimportowania do pliku strony przestrzeni nazw usług sieciowych:

```
<%@ Import Namespace="System.Web.Services" %>
```

Drugi etap polega na dołączeniu treści metody sieciowej do kodu strony. Metoda usługi sieciowej (a dokładnie metoda działająca jak metoda sieciowa) musi zostać oznaczona za pomocą atrybutu `[WebMethod]` — podobnie jak w pliku *.asmx*. Obsługa osadzanych metod usług sieciowych w środowisku ASP.NET AJAX ma również pewne ograniczenia. Oto one:

- Metoda musi być oznaczona za pomocą atrybutu `ScriptMethod`, opisanego w przestrzeni nazw `System.Web.Script.Services`.
- Metoda musi być zadeklarowana jako publiczna (`public`).
- Metoda musi być zadeklarowana jako statyczna (`static`).

Przykład metody spełniającej wszystkie wymienione wymagania został przedstawiony poniżej:

```
<script runat="server">
  [WebMethod]
  [System.Web.Script.Services.ScriptMethod]
  public static float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException( );
    }
    else
    {
      return (float)a / b;
    }
  }
</script>
```

Środowisko ASP.NET AJAX automatycznie wyszukuje wszystkie opisane w ten sposób metody i dołącza je do klasy klienckiej `PageMethods`. Zatem aby wywołać metodę strony, wystarczy posłużyć się zapisem `PageMethods.DivideNumbers()` zgodnie z poniższym przykładem.

```
function callService(f) {
  document.getElementById("c").innerHTML = "";
  PageMethods.DivideNumbers(
  parseInt(f.elements["a"].value),
  parseInt(f.elements["b"].value),
  callComplete,
  callError);
}
```

Ostatnia czynność projektowa pola na włączeniu wywołań do osadzonych metod usług sieciowych. W terminologii ASP.NET AJAX metody te są nazywane „metodami strony” (ang. *page methods*). Ich włączenie wymaga przypisania wartości `true` do właściwości `EnablePageMethods` kontrolki `ScriptManager`:

```
<asp:ScriptManager ID="a1" runat="server" EnablePageMethods="true" />
```

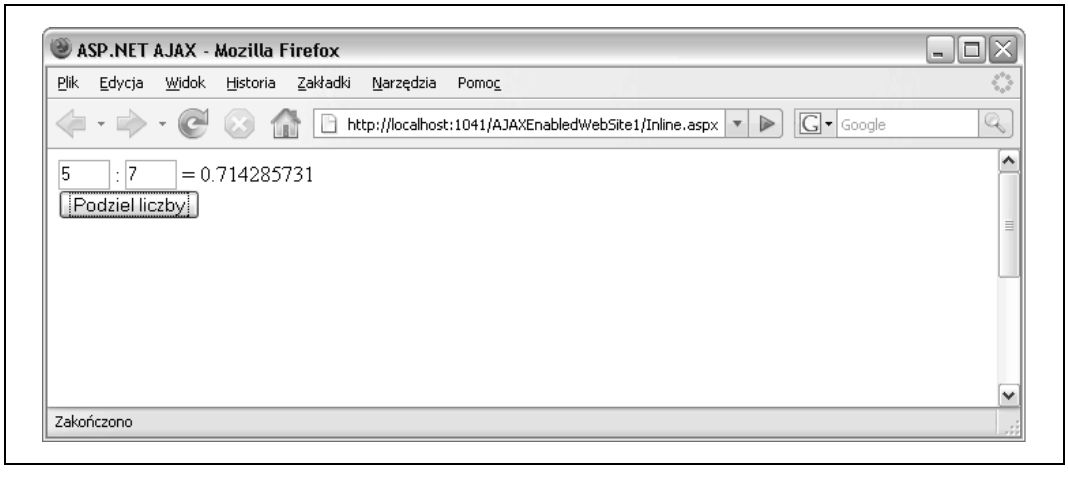
W przykładzie 5.3 został zamieszczony pełen kod strony ASP.NET, w której znajduje się zarówno treść samej strony, jak i metoda usługi sieciowej.

Przykład 5.3. Kod usługi sieciowej i strony ASP.NET AJAX zapisane w jednym pliku

```
Inline.aspx

<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Services" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  [WebMethod]
  [System.Web.Script.Services.ScriptMethod]
  public static float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException();
    }
    else
    {
      return (float)a / b;
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    PageMethods.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result;
  }

  function callError(result) {
    document.getElementById("output").innerHTML =
      "<b>" +
      result.get_exceptionType() +
      "</b>: " +
      result.get_message() +
      "<br />" +
      result.get_stackTrace();
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"
```

```
EnablePageMethods="true">
    </asp:ScriptManager>
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        = <span id="c" style="width: 50px;"></span>
      </nobr>
      <br />
      <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
      <br />
      <div id="output" style="width: 600px; height: 300px;">
      </div>
    </div>
  </form>
</body>
</html>
```

Wynik wyświetlany po załadowaniu strony, wprowadzeniu dwóch wartości i kliknięciu przycisku *Podziel liczby* został pokazany na rysunku 5.2.



Rysunek 5.2. Jeden plik, jedna usługa sieciowa, jedna operacja dzielenia

Przechowywanie informacji o stanie sesji

Usługi sieciowe zyskały sobie miano doskonałej technologii, która nie ma nic wspólnego z aplikacjami sieciowymi. Jednak od kiedy zostały zintegrowane z platformą .NET i witrynami ASP.NET, programiści zyskali możliwość projektowania rozwiązań, które znacznie wykraczają poza funkcje samych usług sieciowych.

Usługi sieciowe platformy .NET pozwalają między innymi na przetwarzanie informacji o stanie sesji. Dane zapisane w sesji są (dzięki środowisku ASP.NET AJAX) udostępniane nawet aplikacjom bazującym na technologii Ajax. Na przykład środowisko ASP.NET AJAX gwarantuje różnym aplikacjom Ajax (uruchomionym na jednym serwerze) dostęp do danych tego samego użytkownika.

Zaimplementowanie opisywanego mechanizmu jest łatwiejsze niż jego omówienie. Za dostęp do danych sesji odpowiada właściwość `EnableSession` atrybutu `[WebMethod]`. Jej przeznaczenie jest takie samo, jak w przypadku metody sieciowej aplikacji .NET.

```
[WebMethod(EnableSession=true)]
```

Po uwzględnieniu właściwości `EnableSession` można bezpośrednio odwoływać się do obiektu `Session` platformy ASP.NET i zapisać lub odczytywać dane. Ponieważ metody sieciowe muszą mieć charakter metod statycznych, konieczne jest zastosowanie odwołania `HttpContext.Current.Session`, a nie po prostu `Session`. Pierwsze z odwołań odnosi się jedynie do obiektów bieżącej instancji klasy `Page`.

W następnym fragmencie skryptu zostały zaprezentowane dwie funkcje. Pierwsza z nich zapisuje bieżącą wartość czasu w sesji. Natomiast druga oblicza różnicę między czasem bieżącym a znacznikiem czasu zapisanym w sesji. Jeśli w sesji nie została zapisana żadna wartość, funkcja zwraca wartość `-1`.

```
[WebMethod(EnableSession = true)]
[System.Web.Script.Services.ScriptMethod]
public static bool SaveTime( )
{
  HttpContext.Current.Session["PageLoaded"] = DateTime.Now;
  return true;
}

[WebMethod(EnableSession = true)]
[System.Web.Script.Services.ScriptMethod]
public static double CalculateDifference( )
{
  if (HttpContext.Current.Session["PageLoaded"] == null) {
    return -1;
  } else {
    DateTime then = (DateTime)HttpContext.Current.Session["PageLoaded"];
    TimeSpan diff = DateTime.Now.Subtract(then);
    return diff.TotalSeconds;
  }
}
```

Powróćmy na chwilę do aplikacji dzielenia dwóch liczb. Do strony zwierającej kod aplikacji zostanie dodana metoda `SaveTime()`, która zapisze wartość czasu, właściwą dla chwili ładowania skryptu. Z kolei w momencie obliczania wyniku dzielenia wyznaczona zostanie różnica między czasem bieżącym a zarejestrowanym wcześniej. W ten sposób będzie można ustalić, ile czasu minęło od pobrania strony do obliczenia wyniku działania (które oczywiście można również wykonać w samym języku JavaScript; celem przykładu jest jednak zademonstrowanie innego rozwiązania).

Kolejny fragment kodu JavaScript odpowiada za wywołanie metody sieciowej (`SaveTime()`), która zarejestruje czas w chwili pobrania strony. Ponieważ w operacji tej nie jest zwracana żadna wartość wynikowa, funkcja zwrotna może być funkcją pustą

```
function pageLoad( ){
  PageMethods.SaveTime(doNothing, doNothing);
}

function doNothing(result) {
  //nic :-)
}
```

Zgodnie z wcześniejszymi założeniami konieczne jest również zdefiniowanie metody (`callService()`), która wywoła metodę `CalculateDifference()` usługi sieciowej. Zamieszczony poniżej kod uwzględnia **dwa** wywołania metod sieciowych. Pierwsze odpowiada za obliczenie różnicy czasu między pobraniem strony a kliknięciem przycisku. Drugie natomiast powoduje wykonanie samego działania matematycznego.

```
function callService(f) {
  document.getElementById("c").innerHTML = "";
  PageMethods.CalculateDifference(
    showDifference,
    callError);
  PageMethods.DivideNumbers(
    parseInt(f.elements["a"].value),
    parseInt(f.elements["b"].value),
    callComplete,
    callError);
}
```

Potrzebny będzie jeszcze pewien kod HTML, który pozwoli na wyświetlenie informacji o czasie. Wykorzystamy do tego celu kontener `<div>`. Należy pamiętać, że wynik o wartości `-1` oznacza, że w sesji nie został zarejestrowany znacznik czasu i w związku z tym nie można obliczyć różnicy czasowej.

```
function showDifference(result) {
  if (result != -1) {
    document.getElementById("output").innerHTML =
    "Formularz był wyświetlany przez " + result + " sekund";
  }
}
```

Kompletny kod strony (treść HTML i skrypt niezbędny do zaimplementowania algorytmu) został przedstawiony w przykładzie 5.4. Wszystkie zmiany w treści zostały odpowiednio wyróżnione. Aby aplikacja działała poprawnie, trzeba pamiętać o dodaniu atrybutu `EnablePageMethods="true"` do kodu kontrolki `ScriptManager`. Brak atrybutu uniemożliwia wywołanie metody strony.

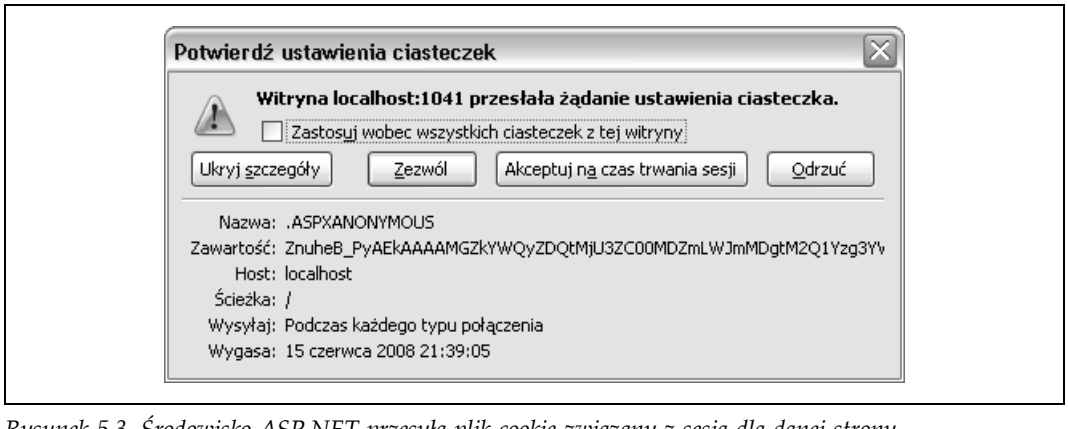
Przykład 5.4. Wykorzystanie sesji w aplikacji ASP.NET AJAX i ASP.NET

```
WebServiceSession.aspx

<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Services" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  [WebMethod(EnableSession = true)]
  [System.Web.Script.Services.ScriptMethod]
  public static bool SaveTime()
  {
    HttpContext.Current.Session["PageLoaded"] = DateTime.Now;
    return true;
  }

  [WebMethod(EnableSession = true)]
  [System.Web.Script.Services.ScriptMethod]
  public static double CalculateDifference()
  {
```

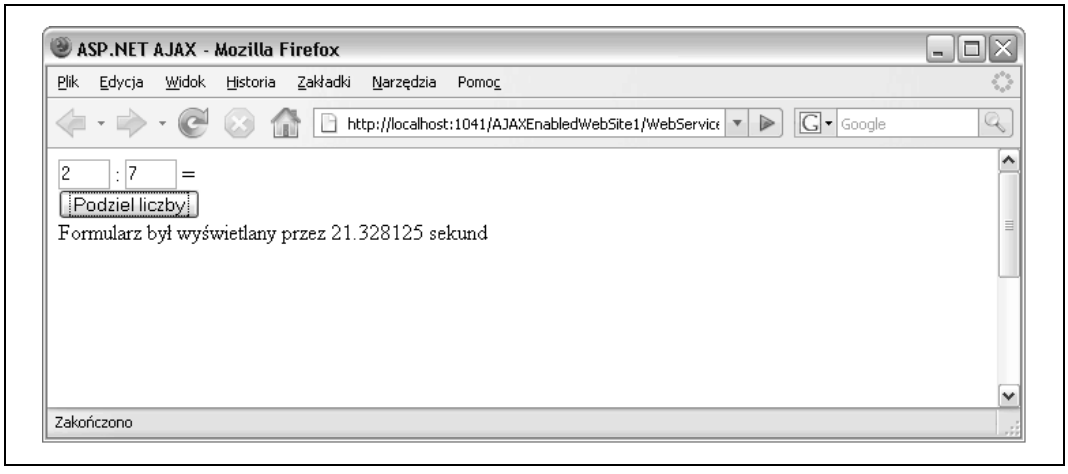
```
      if (HttpContext.Current.Session["PageLoaded"] == null)
      {
      return -1;
    } else {
        DateTime then = (DateTime)HttpContext.Current.Session["PageLoaded"];
      TimeSpan diff = DateTime.Now.Subtract(then);
      return diff.TotalSeconds;
    }
  }

  [WebMethod]
  [System.Web.Script.Services.ScriptMethod] public float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException();
    }
    else
    {
      return (float)a / b;
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function pageLoad() {
    PageMethods.SaveTime(doNothing, doNothing, doNothing);
  }
  function doNothing(result) {
    //nic :-)
  }

  function callService(f) {
    document.getElementById("c").innerHTML = "";
    PageMethods.CalculateDifference(
      showDifference,
      callError);
    PageMethods.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }
function showDifference(result) {
    if (result != -1) {
      document.getElementById("output").innerHTML =
        "Formularz był wyświetlany przez " + result + " sekund";
    }
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result;
  }

  function callError(result) {
    if (result == null) {
      window.alert("Błąd!");
    } else {
```

```
        document.getElementById("output").innerHTML =
          "<b>" +
          result.get_exceptionType() +
          "</b>: " +
          result.get_message() +
          "<br />" +
          result.get_stackTrace();
      }
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server"
        EnablePageMethods="true">
      </asp:ScriptManager>
      <div>
        <nobr>
          <input type="text" id="a" name="a" size="2" />
          :
          <input type="text" id="b" name="b" size="2" />
          = <span id="c" style="width: 50px;"></span>
        </nobr>
        <br />
        <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
        <br />
        <div id="output" style="width: 600px; height: 300px;">
        </div>
      </div>
    </form>
  </body>
  </html>
```

Podczas wykonywania metody `DivideNumbers()` można zauważyć nieco inne działanie przeglądarki niż w poprzednich zadaniach. Po pierwsze, serwer dostarcza plik cookie związany z sesją (o ile w pliku *Web.config* nie została włączona opcja zarządzania sesją bez użycia plików cookie). Jeżeli w przeglądarce została włączona opcja pytania o zezwolenie na przyjęcie pliku cookie, na ekranie powinno się wyświetlić okno, zbliżone do przedstawionego na rysunku 5.3. Druga różnica wiąże się z zachowaniem danych sesji pomiędzy odwołaniami do usługi sieciowej (rysunek 5.4).



Rysunek 5.3. Środowisko ASP.NET przesyła plik cookie związany z sesją dla danej strony



Rysunek 5.4. Wykorzystanie sesji do przechowywania wartości czasu (niezbędnej do obliczenia przerwy między pobraniem strony i wykonaniem działania)

Wymiana złożonych struktur danych między klientem i serwerem

We wcześniejszych przykładach analizowaliśmy jedynie wymianę między serwerem i klientem ciągów tekstowych i wartości typów prostych (liczb, wartości logicznych). Nic jednak nie stoi na przeszkodzie, aby objąć tym mechanizmem również operację dostarczania bardziej złożonych struktur danych. Co prawda język JavaScript nie może konkurować z bogatszymi pod względem liczby typów językami platformy .NET, ale format JSON (opisany w rozdziale 3.) zapewnia podstawową obsługę tablic i obiektów.

Środowisko ASP.NET AJAX jest standardowo wyposażone w mechanizmy serializacji i deserializacji danych JSON. Jeśli więc uwzględnimy je w kodzie usługi sieciowej zaprezentowanej w przykładach 5.1 i 5.2, będziemy mogli udostępnić nową metodę, która za jednym razem zwróci dwie informacje — wynik dzielenia liczb oraz wartość znacznika czasu serwerowego. Aby wdrożyć opisane rozwiązanie, utworzymy w pliku *MathService.asmx* nową klasę, która będzie opisywała zwracany obiekt.

```
public class DivisionData
{
  public float result;
  public string calculationTime;
}
```

Powołanie i zwrócenie obiektu będzie należało do metody przedstawionej poniżej.

```
[WebMethod]
public DivisionData ExtendedDivideNumbers(int a, int b) {
  if (b == 0) {
    throw new DivideByZeroException( );
  } else {
    float res = (float)a / b;
    string stamp = DateTime.Now.ToLongTimeString( );
    DivisionData d = new DivisionData( );
    d.result = res;
```

```
        d.calculationTime = stamp;
        return d;
      }
    }
```

Aby zwracany obiekt był dostępny dla kodu JavaScript, aplikacja ASP.NET AJAX musi go przekształcić (w procesie serializacji) w odpowiedni ciąg JSON. Za użycie właściwej definicji obiektu odpowiada atrybut `GenerateScriptType` (zdefiniowany w przestrzeni nazw `System.Web.Script.Services` [w której zostały zapisane również atrybuty `ScriptService` i `ScriptMethod`]):

```
[System.Web.Script.Services.GenerateScriptType(typeof(DivisionData))]
```

Po stronie serwera nie trzeba wprowadzać więcej zmian. Zaktualizowana treść pliku *MathService.asmx* została zamieszczona w przykładzie 5.5.

Przykład 5.5. Zaktualizowany plik usługi MathService

```
MathService.asmx

<%@ WebService Language="C#" Class="MathService" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

public class DivisionData
{
    public float result;
    public string calculationTime;
}

[WebService(Namespace = "http://hauser-wenz.de/AspNetAJAX/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
[System.Web.Script.Services.GenerateScriptType(typeof(DivisionData))]
public class MathService : System.Web.Services.WebService
{

    [WebMethod]
    public float DivideNumbers(int a, int b)
    {
        if (b == 0)
        {
            throw new DivideByZeroException();
        }
        else
        {
            return (float)a / b;
        }
    }


    [WebMethod]
    public DivisionData ExtendedDivideNumbers(int a, int b)
    {
        if (b == 0)
        {
            throw new DivideByZeroException();
        }
        else
```

```
        {
            float res = (float)a / b;
            string stamp = DateTime.Now.ToLongTimeString();
            DivisionData d = new DivisionData();
            d.result = res;
            d.calculationTime = stamp;
            return d;
        }
    }

}
```

Po stronie klienta deserializacja obiektu `DivisionData` jest realizowana w sposób automatyczny. Obiekt będący wynikiem wywołania usługi sieciowej ma te same właściwości (`result` i `calculationTime`), jakie zostały zdefiniowane w obiekcie `DivisionData`. Instrukcje JavaScript potrzebne do wywołania zmodyfikowanej usługi sieciowej zostały przedstawione w przykładzie 5.6.

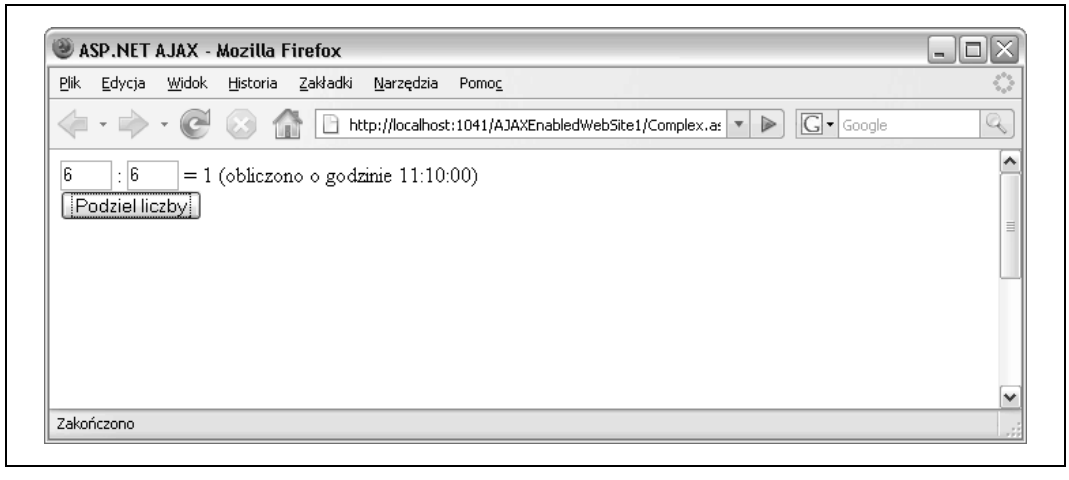
Przykład 5.6. Kod strony pobierającej z metody sieciowej bardziej rozbudowane obiekty

Complex.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    document.getElementById("output").innerHTML = "";
    MathService.ExtendedDivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML =
      result.result +
      " (obliczono o godzinie " +
      result.calculationTime +
      ")";
  }

  function callError(result) {
    document.getElementById("output").innerHTML =
      "<b>" +
      result.get_exceptionType() +
      "</b>: " +
      result.get_message() +
      "<br />" +
      result.get_stackTrace();
  }
  </script>

</head>
```

```
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="MathService.asmx" />
      </Services>
    </asp:ScriptManager>
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        =
        <span id="c" style="width: 50px;"></span>
      </nobr>
      <br />
      <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
      <br />
      <div id="output" style="width: 600px; height: 300px;">
      </div>
    </div>
  </form>
</body>
</html>
```

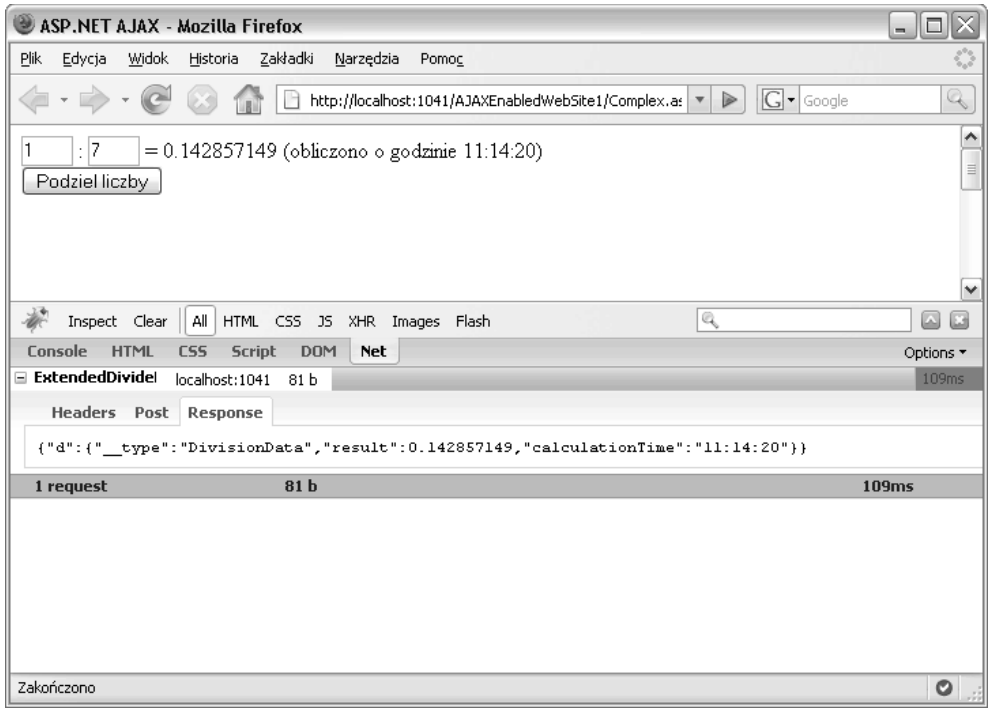
Sposób prezentacji wyniku dzielenia i czasu wygenerowania odpowiedzi został pokazany na rysunku 5.5.



Rysunek 5.5. Wyświetlenie informacji dostarczonych przez serwer

Przechwytując ruch HTTP generowany przez skrypt, możemy sprawdzić, w jaki sposób złożona struktura danych została przekształcona w blok danych JSON (rysunek 5.6).

Opisane do tej pory funkcje środowiska ASP.NET AJAX związane z usługami sieciowymi byłby niezwykle trudne do zaimplementowania za pomocą samego języka JavaScript. Platforma ASP.NET AJAX doskonale integruje się z usługami sieciowymi .NET i stanowi bardzo użyteczny pomost między technologią JavaScript (po stronie klienckiej) i technologią ASP.NET (po stronie serwera).



Rysunek 5.6. Złożona struktura danych po serializacji do formatu JSON

Wykorzystanie usług sieciowych z poziomu skryptu JavaScript

Zapewniane przez środowisko ASP.NET AJAX mechanizmy odwołań do usług sieciowych są niezwykle użyteczne, ponieważ automatycznie realizują wszystkie związane z tą operacją zadania. Zdarzają się jednak sytuacje, w których nie można ich zastosować. Przykładem może być konieczność odwołania się do usługi sieciowej (w tej samej domenie), która nie została napisana dla platformy .NET, lecz opiera się na innych rozwiązaniach serwerowych, takich jak PHP lub Java. Innym powodem bywa również polityka firmy dotycząca stosowania modułów zewnętrznych producentów lub brak akceptacji dla określonej umowy licencyjnej. Ponieważ zakres tematyczny książki wykracza poza samo korzystanie ze środowiska ASP.NET AJAX i obejmuje wszystkie zagadnienia związane z tworzeniem aplikacji Ajax na platformie ASP.NET, omówione zostaną tutaj także zasady wywoływania zdalnych usług sieciowych z poziomu skryptu JavaScript.

Zanim przystąpimy do szczegółowego analizowania stosownych mechanizmów, warto sobie przypomnieć, że model zabezpieczeń języka JavaScript zabrania wykonywania skryptów pochodzących z różnych domen. Oznacza to, że nie można odwołać się do zdalnych witryn za pomocą instrukcji JavaScript (korzystających z obiektu `XMLHttpRequest`).

Istnieją dwie metody programowego wywoływania usług sieciowych w języku JavaScript. Pierwsza z nich polega na zastosowaniu obiektu `XMLHttpRequest`. Natomiast w drugiej zakłada się przygotowanie własnego żądania HTTP SOAP i samodzielną interpretację danych zwracanych przez serwer. Druga metoda jest dość skomplikowana i bardzo podatna na błędy. Znacznie lepszym rozwiązaniem jest wykorzystanie mechanizmów wbudowanych w przeglądarką oraz oficjalnych dodatków do przeglądarki,

Niestety, dwie najpowszechniej stosowane aplikacje — Internet Explorer i Mozilla (czyli Firefox, Epiphany, Camino itd.) — mają zaimplementowane dwie zupełnie różne procedury wywoływania usług sieciowych. W rezultacie programista musi powielać kod zapewniający obsługę każdej z przeglądarek. W końcowej części podrozdziału zostało jednak przedstawione rozwiązanie, które pozwala na połączenie obydwu modeli, a tym samym na opracowanie skryptu (bardziej lub mniej) niezależnego od rodzaju oprogramowania klienckiego.

Usługi sieciowe w przeglądarkach Internet Explorer

Kilka lat temu firma Microsoft rozpoczęła prace na kodem skryptowym, który umożliwiałby wywoływanie usług sieciowych z poziomu samej przeglądarki. Zgodnie z założeniami kod taki powinien powoływać obiekt `XMLHttpRequest`, definiować niezbędne nagłówki HTTP dla żądania SOAP, przygotowywać treść samego żądania, następnie oczekiwać na odpowiedź SOAP i przekształcać wynik do formatu właściwego do dalszego przetwarzania w instrukcjach JavaScript. Ponadto powinien umożliwiać interpretowanie informacji generowanych w języku opisu usług sieciowych (WSDL — ang. *Web Service Description Language*) oraz generowanie lokalnego obiektu pośredniczącego.

Idea nie jest skomplikowana, ale implementacja tak. Ostateczna wersja mechanizmu (wersja 1.0.1.1120) składa się z niemal 2300 wierszy kodu. Niestety, w 2002 roku firma Microsoft przerwała prace nad komponentem komunikacji z usługami sieciowymi. Szkoda, gdyż do dzisiaj jest on wykorzystywany i działa poprawnie. Na szczęście jest jeszcze dostępny w archiwach MSDN pod adresem *http://msdn.microsoft.com/archive/en-us/samples/internet/behaviors/library/webservice/default.asp*.

Aby z niego skorzystać, trzeba pobrać plik *webservice.htc* i zapisać w katalogu, w którym przechowywane są skrypty przykładów. Rozszerzenie *.htc* oznacza kontrolkę HTML (*HTML control*), zwaną też funkcją (ang. *behavior*) przeglądarki Internet Explorer. Do załadowania pliku służy niestandardowa instrukcja stylu CSS, obsługiwana jedynie w aplikacjach Internet Explorer.

```
<div id="WebService" style="behavior:url(webservice.htc);"></div>
```

Nazwa podana jako wartość atrybutu `id` może być wykorzystana w skrypcie w JavaScript zarówno do odwołania do samej kontrolki HTML, jak i do odwołania do usługi sieciowej z nią związanej.

Powiązanie kontrolki z usługą wymaga zdefiniowania odsyłacza do opisu WSDL danej usługi sieciowej. Wykorzystuje się do tego celu metodę `useService()` zapisaną w pliku *.htc*. Konieczne jest również określenie niepowtarzalnego identyfikatora, który umożliwi późniejsze odwoływanie się do danej usługi sieciowej.

```
WebService.useService("MathService.asmx?WSDL", "MathService");
```

Po tych operacjach można wywołać usługę. Kolejność parametrów przekazywanych do metody `callService()` — odpowiadającej za wywołanie usługi sieciowej — różni się od stosowanej w obiektach pośredniczących ASP.NET AJAX. Oto lista tych parametrów:

- referencja do metody zwrotnej,
- nazwa wywoływanej metody sieciowej,
- parametry przekazywane do usługi sieciowej.

Rozwiązanie to nie zapewnia obsługi błędów (w przeciwieństwie do mechanizmów ASP.NET AJAX, które dostarczają wyjątki do skryptu klienckiego).

W przypadku odwołania do usługi `MathService` podzielenie dwóch liczb wymagałoby wykonania następującej instrukcji:

```
WebService.MathService.callService(
  callComplete,
  "DivideNumbers",
  6, 7);
```

Funkcja zwrotna otrzymuje wówczas obiekt, którego atrybut `value` zawiera wynik zwrócony przez usługę sieciową.

```
function callComplete(result) {
  document.getElementsById("c").innerHTML = result.value;
}
```

Pełny kod aplikacji został zamieszczony w przykładzie 5.7.

Przykład 5.7. Wywołanie usługi sieciowej w przeglądarce Internet Explorer

```
MathServiceInternetExplorer.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">

  function callService(f) {
    document.getElementById("c").innerHTML = "";
    WebService.useService("MathService.asmx?WSDL", "MathService");
    WebService.MathService.callService(
      callComplete,
      "DivideNumbers",
      f.elements["a"].value, f.elements["b"].value);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result.value;
  }
  </script>

</head>
<body>
  <div id="WebService" style="behavior:url(webservice.htc);">
  </div>
  <form method="post" onsubmit="return false;">
    <div>
      <nobr>
```

```
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        =
        <span id="c" style="width: 50px;"></span>
      </nobr>
      <br />
      <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
    </div>
  </form>
</body>
</html>
```

Jeśli kontrolka HTML usługi sieciowej nie zostanie zdefiniowana na początku sekcji `<body>`, przeglądarka może wygenerować niepokojące komunikaty o błędach, włącznie z informacją o tym, że obiekt `WebService` nie został zdefiniowany (mimo prawidłowego działania instrukcji `window.alert(WebService)`).

Usługi sieciowe w przeglądarkach Mozilla

Obsługa usług sieciowych została zaimplementowana również w wydawanych ostatnio wersjach przeglądarek Mozilla. Ma ona charakter wbudowanego rozszerzenia. Niestety, komponent odpowiedzialny za komunikację z usługami sieciowymi najwyraźniej nie zyskał szczególnego zainteresowania u osób skupionych wokół projektu Mozilla, choć trzeba przyznać, że poprawnie wykonuje swoje zadanie. W rezultacie nie towarzyszy mu żadna dokumentacja, a informacje na temat jego użycia są często sprzeczne. Rozwiązanie prezentowane w dalszej części punktu pozwala na realizację zadnia, ale wymaga dodania sporej ilości kodu.

Za komunikację z usługami sieciowymi odpowiada klasa `SOAPCall`. Ponieważ opiera się ona na standardzie SOAP 1.1, programista musi zdefiniować nagłówek `SOAPAction` (dostępny w formie właściwości klasy `SOAPClass`) oraz adres URL pliku usługi sieciowej. Oto instrukcje charakterystyczne dla omawianego przykładu:

```
var soapcall = new SOAPCall( );
soapcall.actionURI = "http://hauser-wenz.de/AspNetAJAX/DivideNumbers";
soapcall.transportURI =
"http://localhost:1234/AJAXEnabledWebSite1/MathServiceDocEnc.asmx";
```

Wartość właściwości `transportURI` musi bezwzględnie odpowiadać adresowi URL. Trzeba więc pamiętać o dostosowaniu ciągu URI (szczególnie numeru portu, jeśli do uruchamiania aplikacji jest wykorzystywany serwer testowy środowiska Visual Studio lub Visual Web Developer) do ustawień lokalnego systemu.

Wszystkie parametry przekazywane do usługi są zmiennymi typu `SOAPParameter`. W konstruktorze klasy parametru należy wskazać wartość parametru, a następnie jego nazwę.

```
var p1 = new SOAPParameter(6, "a");
var p2 = new SOAPParameter(7, "b");
```

Bardzo ważne jest wykonanie następnej czynności. Jej ewentualne pominięcie spowoduje, że żądanie SOAP zostanie przesłane do serwera (odebrana zostanie również wartość wyniku), ale nie będą do niego dołączone parametry. W przypadku dzielenia liczb oznaczałoby to niezamierzone wygenerowanie wyjątku dzielenia przez zero (ang. *divie by zero*).

Zadanie polega na osobistym ustaleniu właściwego kodowania dla wartości liczbowych. W tym celu należy załadować odpowiednią przestrzeń nazw, która obejmuje typ SOAP `integer`. Następnie trzeba określić wartość właściwości `schemaType` wszystkich parametrów, które powinny być przekazane do usługi sieciowej, przypisując im wygenerowane typy danych. Kod realizujący opisane zadania został zamieszczony poniżej.

```
var senc = new SOAPEncoding( );
assenc = senc.getAssociatedEncoding(
  "http://schemas.xmlsoap.org/soap/encoding/",
  false);
var scoll = assenc.schemaCollection;
var stype = scoll.getType(
  "integer",
  "http://www.w3.org/2001/XMLSchema");
p1.schemaType = stype;
p2.schemaType = stype;
```

Kolejna czynność polega na przygotowaniu wywołania usługi sieciowej. Służy do tego metoda `encode()`, wymagająca przekazania do niej kilku parametrów, zgodnie z poniższym przykładem.

```
soapcall.encode(
  0,                                  //wartość domyślna dla protokołu SOAP 1.1
  "DivideNumbers",                    //nazwa metody sieciowej
  "http://hauser-wenz.de/AspNetAJAX/",  //przestrzeń nazw
  0,                                  //liczba dodatkowych nagłówków
  new Array( ),                       //dodatkowe nagłówki
  2,                                  //liczba parametrów
  new Array(p1, p2)                   //parametry
);
```

W końcu można wywołać usługę sieciową (w sposób asynchroniczny), wykonując metodę `asyncInvoke()`. Parametrem metody jest referencja funkcji zwrotnej.

```
soapcall.asyncInvoke(callComplete);
```

Funkcja zwrotna otrzymuje trzy parametry:

- dokument XML będący wynikiem wywołania usługi sieciowej,
- obiekt `SOAPCall` (gdyby konieczne było przeanalizowanie nagłówków SOAP),
- kod statusowy HTTP dla wywołania.

Ostatni etap procedury sprowadza się do wyodrębnienia danych wynikowych z treści dostarczonego dokumentu XML. Przyjrzyjmy się zatem odpowiedzi XML zwracanej po odwołaniu się do usługi `MathService` — dane te można pozyskać za pomocą programu Fiddler (dla systemu Windows) (*http://www.fiddlertool.com/fiddler*) lub rozszerzenia przeglądarki Mozilla Live HTTP Headers (*http://livehttpheaders.mozdev.org/*):

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap
.org/soap/envelope/">
  <soap:Body>
     <DivideNumbersResponse xmlns="http://hauser-wenz.de/AspNetAJAX/">
      <DivideNumbersResult>0.857142866</DivideNumbersResult>
    </DivideNumbersResponse>
  </soap:Body>
</soap:Envelope>
```

Więcej informacji na temat sposobu analizowania żądań HTTP (wysyłanych przez aplikacje Ajax) oraz debugowania aplikacji Ajax znajduje się w dodatku A.

Analizując dane w formacie XML, nietrudno zauważyć, że wyodrębnienie wartości `0.857142866` wymaga przeprowadzenia następujących operacji:

- odwołania się do właściwości `body`, która zapewni dostęp do elementu `<soap:Body>`;
- odwołania się do właściwości `firstChild`, która zapewni dostęp do elementu `<DivideNumbersResponse>`;
- ponownego wykorzystania właściwości `firstChild` do pobrania elementu `<DivideNumbersResult>`;
- wykorzystania po raz trzeci właściwości `firstChild`, aby uzyskać dostęp do węzła tekstowego w elemencie `<DivideNumbersResult>`;
- wykorzystania właściwości `data` do pobrania ciągu zapisanego w węźle tekstowym.

Treść skryptu JavaScript niezbędnego do wyodrębnienia wyniku z dokumentu dostarczonego przez usługę sieciową została zamieszczona poniżej.

```
function callComplete(result, soapcall, status) {
  document.getElementById("c").innerHTML =
    result.body.firstChild.firstChild.firstChild.data;
}
```

Łącząc wszystkie opisane fragmenty skryptów, uzyskujemy kod przedstawiony w przykładzie 5.8. Trzeba jednak pamiętać, że aplikacja będzie działała zgodnie z oczekiwaniami tylko wtedy, gdy system będzie miał dostęp do internetu — przeglądarki Mozilla muszą mieć dostęp do informacji na temat schematów SOAP.

Przykład 5.8. Wywołanie usługi sieciowej w przeglądarce Mozilla

```
MathServiceMozilla.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
    function callService(f) {
    document.getElementById("c").innerHTML = "";
    var soapcall = new SOAPCall( );
    soapcall.actionURI = "http://hauser-wenz.de/AspNetAJAX/DivideNumbers";

soapcall.transportURI="http://localhost:1041/AJAXEnabledWebSite1/MathService.asmx";
    var p1 = new SOAPParameter(parseInt(f.elements["a"].value), "a");
    var p2 = new SOAPParameter(parseInt(f.elements["b"].value), "b");

    var senc = new SOAPEncoding( );
    assenc = senc.getAssociatedEncoding(
      "http://schemas.xmlsoap.org/soap/encoding/",
      false);
    var scoll = assenc.schemaCollection;
    var stype = scoll.getType(
      "integer",
```

```
      "http://www.w3.org/2001/XMLSchema");
    p1.schemaType = stype;
    p2.schemaType = stype;

    soapcall.encode(
      0,                                  //wartość domyślna dla protokołu SOAP 1.1
      "DivideNumbers",                    //nazwa metody sieciowej
      "http://hauser-wenz.de/AspNetAJAX/",  //przestrzeń nazw
      0,                                  //liczba dodatkowych nagłówków
      new Array( ),                       //dodatkowe nagłówki
      2,                                  //liczba parametrów
      new Array(p1, p2)                   //parametry
    );
    soapcall.asyncInvoke(callComplete);
  }
  function callComplete(result, soapcall, status) {
    document.getElementById("c").innerHTML =
      result.body.firstChild.firstChild.firstChild.data;
  }
  </script>

</head>
<body>
  <form method="post" onsubmit="return false;">
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        =
        <span id="c" style="width: 50px;"></span>
      </nobr>
      <br />
      <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
    </div>
  </form>
</body>
</html>
```

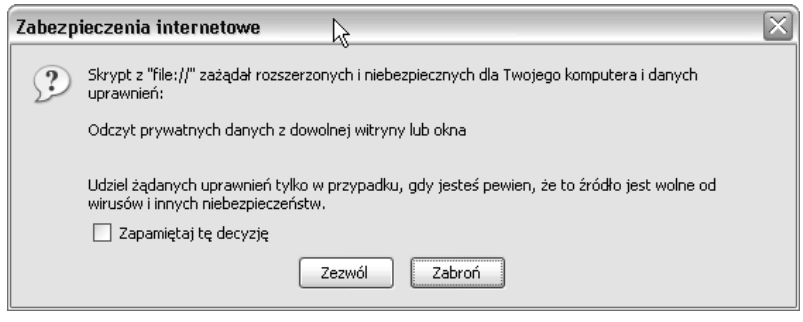
Usługi sieciowe w obydwu przeglądarkach

Przegląd technik korzystania z usług sieciowych z poziomu skryptu JavaScript w przeglądarkach Internet Explorer i Mozilla zakończymy rozwiązaniem, które łączy obydwie metody na jednej stronie. W tym celu musimy przede wszystkim ustalić, w jaki sposób rozpoznawany będzie rodzaj przeglądarki. Zgodnie z informacjami zamieszczonymi w rozdziale 2, najkorzystniejsze wydaje się sprawdzenie zestawu funkcji przeglądarki, a nie ich typu. Zasada ta została wykorzystana podczas opracowywania kodu z przykładu 5.9 (jej opis znajduje się w rozdziale 2, w części dotyczącej powoływania obiektu `XMLHttpRequest`). Rozwiązanie polega na utworzeniu obiektu właściwego dla jednej przeglądarki. Jeśli to się uda, dalsza część kodu zostanie wykonana zgodnie z założeniami. W przeciwnym przypadku wykorzystane zostaną instrukcje właściwe dla drugiej przeglądarki. Poszczególne wywołania zostały zapisane w dwóch zagnieżdżonych konstrukcjach `try … catch`.

Dostęp do zdalnych usług sieciowych w przeglądarkach Mozilla

Model zabezpieczeń przeglądarki Mozilla umożliwia odwołania do zdalnych usług sieciowych. Wykonanie skryptu wiąże się jednak z wyświetleniem okna dialogowego, w którym użytkownik musi zezwolić na taką operację (rysunek 5.7). Wymagane jest w tym przypadku uprawnienie `UniversalBrowserRead`, oznaczające, że przeglądarka może pobierać dane z dowolnego serwera (włączając w to zarówno serwery zdalne, jak i lokalny system plików).

```
netscape.security.PrivilegeManager.enablePrivilege(
  "UniversalBrowserRead");
```



Rysunek 5.7. Żądanie zwiększenia poziomu uprawnień w przeglądarce Firefox

Domyślna konfiguracja przeglądarek Mozilla i Firefox (a także kilku innych) powoduje nadanie wspomnianego uprawnienia tylko w dostępie do plików lokalnych (z definicją protokołu `file://`). Mechanizm ten znajduje więc zastosowanie głównie w aplikacjach intranetowych. Wygląd okna dialogowego z żądaniem zwiększenia poziomu uprawnień został pokazany na rysunku 5.7.

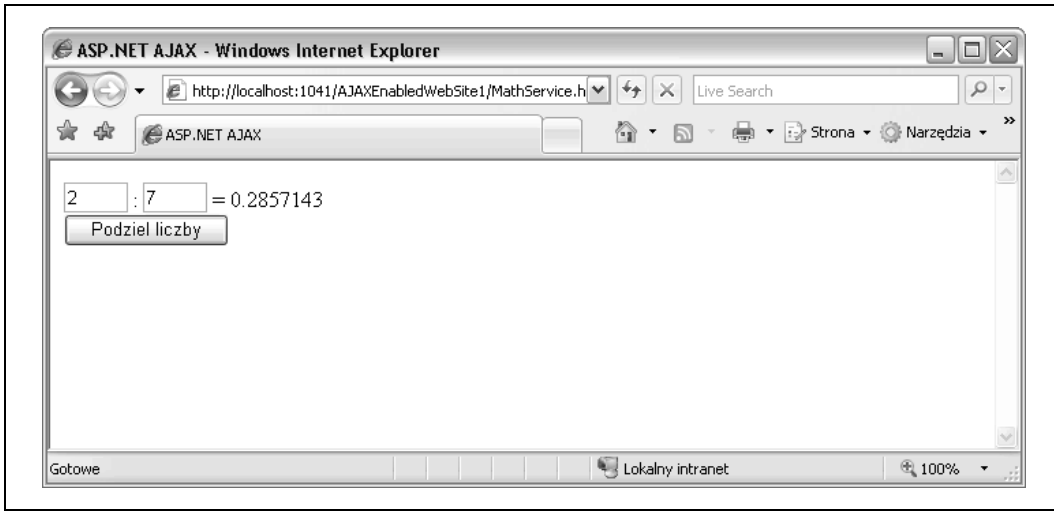
Przykład 5.9. Wywołanie usługi sieciowej w dowolnej z przeglądarek Internet Explorer i Mozilla

```
MathService.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
    function callService(f) {
    document.getElementById("c").innerHTML = "";
    try {
      WebService.useService("MathService.asmx?WSDL", "MathService");
      WebService.MathService.callService(
        callComplete,
        "DivideNumbers",
        parseInt(f.elements["a"].value), parseInt(f.elements["b"].value));
    } catch (e) {
      try {
        var soapcall = new SOAPCall( );
        soapcall.actionURI = "http://hauser-wenz.de/AspNetAJAX/DivideNumbers";
```

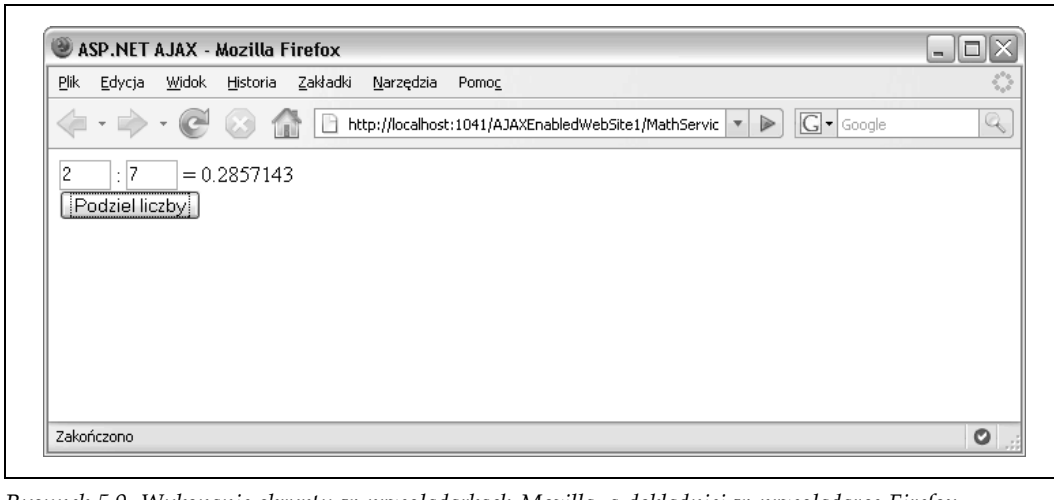
```
        soapcall.transportURI = "http://localhost:1041/AJAXEnabledWebSite1/
        MathService.asmx";

        var p1 = new SOAPParameter(parseInt(f.elements["a"].value), "a");
        var p2 = new SOAPParameter(parseInt(f.elements["b"].value), "b");

        var senc = new SOAPEncoding( );
        assenc = senc.getAssociatedEncoding(
          "http://schemas.xmlsoap.org/soap/encoding/",
          false);
        var scoll = assenc.schemaCollection;
        var stype = scoll.getType(
          "integer",
          "http://www.w3.org/2001/XMLSchema");
        p1.schemaType = stype;
        p2.schemaType = stype;

        soapcall.encode(
          0,                                  //wartość domyślna dla protokołu SOAP 1.1
          "DivideNumbers",                     //nazwa metody sieciowej
          "http://hauser-wenz.de/AspNetAJAX/",  //przestrzeń nazw
          0,                                   //liczba dodatkowych nagłówków
          new Array( ),                        //dodatkowe nagłówki
          2,                                   //liczba parametrów
          new Array(p1, p2)                    //parametry
        );
        soapcall.asyncInvoke(callComplete);
      } catch (e) {
        window.alert("Twoja przeglądarka nie jest obsługiwana.");
      }
    }
  }

  function callComplete(result, soapcall, status) {
    if (result.value != null) {
      document.getElementById("c").innerHTML = result.value;
    } else {
      document.getElementById("c").innerHTML =
      result.body.firstChild.firstChild.firstChild.data;
    }
  }
</script>

</head>
<body>
  <div id="WebService" style="behavior: url(webservice.htc);">
  </div>
  <form method="post" onsubmit="return false;">
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        = <span id="c" style="width: 50px;" ></span>
      </nobr>
      <br />
      <input type="button" value="Podziel liczby" onclick="callService(this.form);" />
    </div>
  </form>
</body>
</html>
```

W przykładzie 5.9 zostały zamieszczone wszystkie znaczniki i instrukcje skryptowe, które pozwalają na wykonanie zadania. Nie zmienia to faktu, że przed wdrożeniem aplikacji trzeba ją przetestować również w innych przeglądarkach. Należy również pamiętać o przypisaniu właściwości `soapcall.transportURI` poprawnego adresu URL witryny (włącznie z numerem portu, jeśli jest to konieczne).

Jak można się przekonać, analizując rysunki 5.8 i 5.9, opracowana aplikacja działa poprawnie w dwóch najpowszechniej stosowanych przeglądarkach.



Rysunek 5.8. Wykonanie skryptu w przeglądarce Internet Explorer



Rysunek 5.9. Wykonanie skryptu w przeglądarkach Mozilla, a dokładniej w przeglądarce Firefox

Pozostaje jeszcze tylko problem ustalenia, czy uzyskany efekt jest wart dodatkowej pracy. Czy rzeczywiście przygotowanie niezbyt uniwersalnego rozwiązania do wywoływania usług sieciowych można określić jako użyteczne? Witryny bazujące na platformie ASP.NET realizują takie zadania bezproblemowo z wykorzystaniem kodu ASP.NET AJAX. Biorąc pod uwagę łatwość wdrażania aplikacji ASP.NET AJAX, metoda opisana w tym punkcie powinna być postrzegana jako „ostatnia deska ratunku" (szczególnie jeśli uwzględnimy wstrzymanie prac nad rozwojem usług sieciowych w oprogramowaniu Mozilla).

Podsumowanie

W niniejszym rozdziale zostało opisanych kilka sposobów wykorzystania usług sieciowych. W pierwszym podrozdziale zostały omówione zagadnienia związane z obsługą błędów i przechowywaniem danych w sesji. Kolejne punkty zawierają przykłady wymiany złożonych danych między aplikacjami klienckimi i usługami sieciowymi, a także przykłady rozwiązań umożliwiających wywoływanie z poziomu skryptu JavaScript usług sieciowych udostępnionych na serwerach innych niż ASP.NET.

Do dalszego czytania

http://msdn.microsoft.com/archive/en-us/samples/internet/behaviors/library/webservice/default.asp
: Wcześniejsze wersje kontrolki *webservice.htc*.

http://ajax.asp.net/docs/tutorials/ASPNETAJAXWebServicesTutorials.aspx
: Przewodnik dotyczący usług sieciowych w dokumentacji Microsoft ASP.NET AJAX.