Drukuj Zamknij

Podstawy Web Services - własne usługi oraz Google Web API

2003-12-11 22:24:52

Michał Malczewski - Katedra Systemów Informacyjnych, Polsko-Japońska Wyższa Szkoła Technik Komputerowych w Warszawie

Słowem wstępu

Web Services są bardzo ważnym elementem platformy .NET. Co to tak naprawdę jest? Mówiąc najprościej, jest to mechanizm służący udostępnianiu zaimplementowanej logiki biznesowej konkretnego serwera (i danych, które posiada) innym aplikacjom sieci Web. Wykorzystywane są przy tym standardowe protokoły (HTTP, XML, SOAP), dzięki czemu Twoja aplikacja napisana w środowisku .NET dostępna będzie również dla wielu innych platform, jak Java, PHP, Python i innych.

Odrobina teorii

Napisanie prostej aplikacji działającej jako Web Service jest naprawdę proste – prosty „Hello World" jest dołączany przez Visual Studio .NET do każdego projektu tego typu (jest domyślnie zakomentowany). Warto jednak mieć podstawowe informacje o mechanizmach, które wykorzystamy tworząc naszą aplikację.

SOAP – Simple Object Access Protocol. Jest to protokół definiujący sposób wymiany danych, wykorzystywany przez Web Services. W związku z tym, że jest zdefiniowany na podstawie języka XML, jest niezależny zarówno od języka, jak i od platformy, na której będzie wykorzystany.

WSDL – Web Services Description Language. Zgodnie z nazwą, standard wykorzystywany jest do opisu Web Service. Opisuje zarówno zbiór dostępnych serwisów na serwerze, jak i metod, które udostępniają. Dzięki niemu klient wie co dany serwis mu umożliwia, jakie stawia wymagania i w jaki sposób należy się z nim komunikować.

Do stworzenia pierwszej aplikacji wystarczy nam wiedza na temat protokołów, które będzie wykorzystywać. Oczywiście, chcąc dalej rozwijać temat, należy dokładniej się z nim zapoznać. Jest to bardzo istotne dla późniejszego rozwoju aplikacji – chociażby zabezpieczanie Web Services bardzo wygodnie jest wykonywać w oparciu o dodatkowe nagłówki SOAP.

Pierwsza aplikacja

Przejdźmy więc do praktyki. Na początku utworzymy prosty Web Service, który policzy sumę 2 liczb. Stwórzmy więc nowy projekt (ASP. NET Web Service), a żeby zachować porządek zmieńmy nazwę pliku *Service1.asmx* na *MyMath.asmx*. Jest to aplikacja sieci Web, więc została utworzona na dysku wirtualnym serwera IIS, a jej źródła i binaria znajdują się w katalogu przeznaczonym na aplikacje serwera IIS (domyślnie c:\inetpub\wwwroot). Można od razu zaobserwować, że aplikacje Web Services noszą rozszerzenie .asmx. Otwórzmy plik *MyMath.asmx*.

Widać na początku, że nasza klasa Service1 (zmieńmy od razu jej nazwę na MyMath) dziedziczy po System.Web.Services.WebService. Widać też wspomniany wcześniej HelloWorld, odkomentujmy go – zostanie wykorzystany w późniejszym przykładzie.

Metody mające być udostępnione jako Web Service posiadają atrybut [WebMethod] przy swojej sygnaturze, co widać na przykładzie HelloWorld. Stwórzmy jednak naszą metodę wykonującą dodawanie. Zwraca obiekt typu int, a jej parametrami są 2 inne obiekty typu int. Uwzględniając to, co powiedzieliśmy na temat metod Web Services może ona wyglądać następująco:

```
[WebMethod]
public int addNumbers(int argA, int argB)
{
     return argA + argB;
}
```

Sprawdźmy, czy to działa – uruchommy aplikację klawiszem F5. Otwiera się przeglądarka internetowa i wyświetla nasz Web Service. Można w tym oknie znaleźć kilka interesujących informacji.

Po pierwsze, lista naszych metod. Widać, że dostępne są HelloWorld oraz AddNumbers. Po kliknięciu na odpowiedni link możemy przetestować ich działanie. Wyświetlany jest od razu komunikat SOAP, który będzie przesyłany w trakcie uruchamiania metod. Obydwie metody nie wymagają ani nie zwracają typów złożonych (jak tablice), można więc od razu sprawdzić ich działanie (HelloWorld zwróci string, a addNumbers po przyjęciu parametrów zwróci ich sumę). W przypadku bardziej złożonych metod nie jest możliwe ich wywoływanie z tego miejsca (w jaki sposób przekazalibyśmy tablicę?).

Drugą sprawą, która powinna nas zainteresować jest link Opis Usług (ang. Service Description) na głównej stronie opisu serwisu. Jak łatwo zauważyć, jest to ten sam adres, jednak z dodatkowym parametrem w

URL: ?wsdl. Jest to najprostszy sposób na uzyskanie opisu serwisu. Polecam przyjrzenie się opisowi. Oczywiście, po wcześniejszym przeczytaniu jakiegoś opisu WSDL wyświetlane informacje będą bardziej zrozumiałe, jednak na początek są wystarczająco mówiące. Opis ten będzie nam potrzebny w trakcie przygotowywania aplikacji klienckiej.

Pierwszy klient

Zajmiemy się teraz utworzeniem aplikacji klienckiej, która wykorzysta nasz Web Service. Dodajmy do solucji nowy projekt: będzie to aplikacja konsolowa. Teraz bardzo ważna sprawa – umożliwienie klientowi korzystania z naszego serwisu. W widoku *Solution Explorer* kliknijmy na nasz projekt zawierający Web Service. W oknie *Properties* znajduje się teraz pole *Project Folder*. Skopiujmy jego zawartość (jest to adres to Web Service który wykorzystamy). W *Solution Explorer* kliknijmy prawym klawiszem na elemencie *References* i wybierzmy opcję *Add Web Reference*. W polu URL wklejmy adres serwisu który chcemy wykorzystać i dodajmy nazwę pliku (*MyMath.asmx*) oraz suffix *?wsdl* (już wiemy co jego użycie spowoduje). W polu *Reference Name* podajmy *myMathService*.

Et voila – możemy wykorzystać nasz serwis w aplikacji konsolowej. Dzięki referencji możemy odnosić się do obiektu, do którego referencja się odnosi. Wykorzystajmy to więc – wywołajmy nasze metody i wypiszmy na ekran wyniki:

```
myMathService.MyMath myMath = new myMathService.MyMath();
// wyswietlamy powitanie
Console.WriteLine(myMath.HelloWorld());

// dodajemy 2 liczby
int argA = 2;
int argB = 2;
int result = myMath.AddNumbers(argA, argB);
Console.WriteLine("{0} + {1} = {2}", argA, argB, result);
Console.ReadLine();
```

Wystarczy ustawić aplikację kliencką jako projekt startowy i wcisnąć F5.

Wykorzystujemy „cudzy" Web Service

Aby udowodnić, że Web Services to nie tylko pisanie programów na własnym komputerze, napiszemy prostą aplikację ASP przeszukującą Internet za pomocą wyszukiwarki Google.

Wykorzystamy przy tym Google Web API – Web Service udostępniony przez Google. Google Web API umożliwia wykorzystanie wyszukiwarki na wiele sposobów. W przykładzie zajmiemy się prostym wyszukiwaniem.

Aby w ogóle móc z Google Web API skorzystać, należy dokonać rejestracji na stronie http://www.google.com/apis/. Rejestracja jest bezpłatna, szybka i nie nastręcza żadnych problemów. Będzie potrzebny klucz który Google przyśle na wskazany adres e-mail.

Utwórzmy nowy projekt ASP.NET i dodajmy referencję do Google Web API: http://api.google.com/GoogleSearch.wsdl (w trakcie dodawania referencji wymagane jest połączenie z Internetem). Dla spójności z resztą artykułu nazwijmy referencję *Google*.

Do formy Web dodajmy 3 pola: TextBox do wprowadzania zapytania, Button do wywołania zapytania, oraz DataGrid do wyświetlenia wyników. Nazwijmy je odpowiednio *queryStringTestBox*, *submitQueryButton* oraz *resultsGrid*. Utwórzmy również pole *googleKey*, w którym przechowywać będziemy klucz rejestracyjny otrzymany od Google w poprzednim kroku.

Do obsługi zapytań służy klasa *GoogleSearchService*. Między innymi posiada metodę najbardziej nas w tej chwili interesującą – *doGoogleSearch*. Jej wynikiem jest obiekt klasy *GoogleSearchResult*, który oczywiście reprezentuje wyniki zapytania. Metoda *doGoogleSearch* przyjmuje kilka parametrów dokładnie specyfikujących zapytanie, nas jednak najbardziej interesować będzie tylko kilka z nich. Dla wygody upakujemy tę funkcjonalność w metodę. Pod koniec odwołuje się ona do metody *fillGrid*, jej opis znajduje się poniżej. Zajmijmy się więc wykonaniem zapytania:

```
private void performSearch()
{
    Google.GoogleSearchService googleSearch = new Google.GoogleSearchService();

    string searchQuery = queryStringTestBox.Text;
    int startIndex = getSessionSearchStartIndex();

  Google.GoogleSearchResult searchResults = googleSearch.doGoogleSearch(
        googleKey, searchQuery, startIndex, 10, false, "", false, "", "", ""
    );

    Session["estimatedTotalResultsCount"] =
```

```
        searchResults.estimatedTotalResultsCount;


    fillGrid(searchResults.resultElements);
}
```

Jak widać, pierwszy parametr metody *doGoogleSearch* to klucz umożliwiający korzystanie z serwisu. Następnie podajemy treść pytania (zawartość utworzonego przed chwilą pola TextBox), indeks początkowy i maksymalną ilość wyników na stronie. Indeks początkowy określa początek przedziału, z którego będą zwrócone wyniki. Maksymalna liczba wyników na stronie nie może przekroczyć 10 – w przeciwnym wypadku zostanie wysłany wyjątek informujący klienta o tym fakcie. Następne pola dotyczą ustawienia dodatkowych filtrów, ustawiamy więc boolowskie pola na *false*, a tekstowym przypisujemy pusty ciąg znaków.

Kod przedstawiony powyżej wykona zapytanie, pozostaje nam jeszcze przedstawić wyniki. Wykorzystamy w tym celu utworzoną wcześniej kontrolkę *DataGrid*, którą nazwaliśmy *resultGrid*. Dane, które będą wyświetlone w rezultacie umieścimy w *DataTable*. Obiekt ten reprezentuje dane w formie tabelarycznej – każdy wynik stanowić będzie odrębny wiersz tabeli. Występować będzie tylko jedna kolumna wyświetlająca nasz wynik (nadajmy jej tytuł „Search Results”).

```
private void fillGrid(Google.ResultElement[] results)
{
     DataTable dataTable = new DataTable();
     dataTable.Columns.Add(new DataColumn("Results"));

     for(int i = 0; i < results.Length; i++)
     {
          string parsedResult =
               "<b>" + results[i].title + "</b><br>\n";

          // niektore podsumowania sa puste, nie wyswietlamy ich
          if(results[i].summary.Length > 1)
               parsedResult += results[i].summary + "<br>\n" ;

          // dodajemy link
          parsedResult +=
               "<a href=\"" + results[i].URL +
               "\" target=\"_blank\">" + results[i].URL + "</a>";

          // utworzenie nowego wiersza
          DataRow row = dataTable.NewRow();

          // wstawienie wyniku do wiersza
          row[0] = parsedResult;

          //dodanie wiersza do tabeli
          dataTable.Rows.Add(row);
     }

     // ustanowienie zrodla danych
     resultsGrid.DataSource = dataTable;

     // synchronizacja danych
     resultsGrid.DataBind();
}
```

Pierwsza wersja aplikacji gotowa. Wystarczy obsłużyć zdarzenie onClick naszego *submitQueryButton* dodając wywołanie metody *performSearch* i wyniki powędrują na ekran.

Obsługa dużej ilości wyników

Jak powiedzieliśmy wcześniej, jednorazowo można pobrać do 10 wyników z bazy. Należy więc umożliwić nawigację pomiędzy kolejnymi rekordami. Trzeba pamiętać pomiędzy kolejnymi wyświetleniami strony które wyniki są wyświetlane oraz ilość wszystkich wyników. W aplikacjach webowych służą do takich celów sesje. Obsługa sesji jest wyjątkowo prosta. Obiekt *Session* jest niejawnie deklarowaną tablicą do której możemy wrzucać elementy, np.:

```
Session[„moja zmienna”] = 1234567;
```

W naszej aplikacji dodamy jeszcze 2 przyciski – przewijanie w tył i przewijanie w przód (mogą być początkowo ukryte – atrybut *Visibility* ustawiamy na *false*). Do naszych zmiennych sesyjnych będziemy odwoływać się w kilku miejscach w programie. Ponadto, jeżeli jest to pierwsze odwołanie do zmiennej sesyjnej należy nadać jej odpowiednią początkową wartość. Dlatego zróbmy dla tych wartości odpowiednie właściwości klasy: sessionSearchStartIndex i sessionEstimatedTotalResultsCount.

```
private int sessionSearchStartIndex
{
     get
     {
          return
               Session["searchStartIndex"] != null
               ? (int)Session["searchStartIndex"]: 0;
     }
     set
```

```
    {
        Session["searchStartIndex"] = value;
    }
}
```

Tak samo wyglądać będzie dostęp do drugiej zmiennej. Jak już powiedzieliśmy wcześniej, jednym z parametrów metody *doGoogleSearch* jest offset określający początek podzbioru wszystkich rekordów odpowiadających zapytaniu. Zmodyfikujmy więc jej wywołanie:

```
Google.GoogleSearchResult searchResults = googleSearch.doGoogleSearch(
    googleKey, searchQuery, sessionSearchStartIndex, 10,
    false, "", false, "", "", ""
);
```

Teraz pozostaje tylko obsługa przycisków przewijających rekordy. Jeżeli wyników nie jest dostatecznie dużo, lub offset dochodzi do końca zbioru wyników, przycisk przewijający do przodu jest zbędny. Podobna sytuacja ma miejsce z przewijaniem do tyłu. Ponadto chcielibyśmy wyświetlać informację o rezultatach – ich ogólną liczbę i aktualną pozycję (wykorzystamy kontrolkę Label, dodajmy takową do formy). Stwórzmy więc metodę, która będzie odpowiadać za te rzeczy.

```
private void handleResultCount()
{
    if(sessionEstimatedTotalResultsCount > 0)
    {
        this.resultsCountDescription.Text =
            "Wniki " + sessionSearchStartIndex.ToString() +
            " do " + (sessionSearchStartIndex + 10) + " z ok. "
            + sessionEstimatedTotalResultsCount;
    }
    else
    {
        this.resultsCountDescription.Text = "Nic nie znaleziono";
    }

    prevResultsButton.Visible = (sessionSearchStartIndex > 0 ? true : false);

    nextResultsButton.Visible = (
        (sessionSearchStartIndex < sessionEstimatedTotalResultsCount)
        && sessionEstimatedTotalResultsCount > 10
        ? true : false
    );
}
```

Metodę należy wywoływać za każdym razem, gdy następuje szukanie w Googlu, celowym będzie więc wywoływanie jej z metody *performSearch*.

Ostatnim krokiem jest obsługa nowych przycisków. Wszystko co trzeba wykonać, to odpowiednio zmniejszenie lub zwiększenie offsetu i wywołanie szukania, obsługa kliknięcia może więc wyglądać następująco:

```
private void prevResultsButton_Click(object sender, System.EventArgs e)
{
    sessionSearchStartIndex = sessionSearchStartIndex - 10;
    performSearch();
}
```

Gotowa aplikacja

Nasza wyszukiwarka już działa. W razie potrzeby źródło programu można znaleźć w załączniku. Pamiętajmy jednak, że aplikacja nie zadziała bez podania klucza rejestracyjnego. Jak już jednak wspominałem, nie nastręcza to żadnych problemów, a podstawowa licencja umożliwia wykonanie 1000 zapytań dziennie, co z pewnością wystarczy na przetestowanie programu.

Drukuj Zamknij