

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

ASP.NET Ajax. Intensywny trening

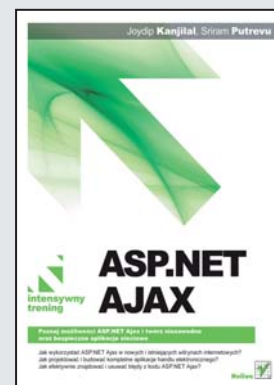
Autor: Joydip Kanjilal, Sriram Putrevu

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-246-2082-1

Tytuł oryginału: [Sams Teach Yourself ASP.NET with Ajax in 24 Hours](#)

Format: 170x230, stron: 440



ASP.NET Ajax. Intensywny trening

Poznaj możliwości ASP.NET Ajax i twórz niezawodne oraz bezpieczne aplikacje sieciowe

- Jak wykorzystać ASP.NET Ajax w nowych i istniejących witrynach internetowych?
- Jak projektować i budować kompletne aplikacje handlu elektronicznego?
- Jak efektywnie znajdować i usuwać błędy z kodu ASP.NET Ajax?

Wraz z pojawieniem się technologii Ajax skończyły się żmudne poszukiwania programistów, pragnących znaleźć technologię umożliwiającą szybsze wizualizowanie stron, krótszy czas odpowiedzi oraz asynchroniczne przetwarzanie. Architektura ASP.NET Ajax została zaprojektowana w taki sposób, żeby działać zarówno po stronie klienta, jak i serwera. Wyposażenie jej w kilka bibliotek sprawia, że możliwe jest tworzenie komponentów odpowiedzialnych m.in. za usługi sieciowe i aplikacyjne, różne podstawowe usługi dotyczące serializacji oraz rozszerzenia podstawowych klas JavaScript.

Książka „ASP.NET Ajax. Intensywny trening” zawiera zestaw prostych i praktycznych instrukcji, z których każda poprowadzi cię krok po kroku przez tajniki stosowania Ajaksa w aplikacjach internetowych. Z tego podręcznika dowiesz się, jak maksymalnie wykorzystywać możliwości bibliotek Microsoft Ajax Library i Ajax Server Extensions. Nauczysz się korzystać ze wszystkich najważniejszych rozszerzeń serwerowych, stosować techniki programowania po stronie klienta, maksymalizować wydajność usług sieciowych, a także projektować i budować kompletne aplikacje handlu internetowego.

- Architektura ASP.NET Ajax
- Obiekt XMLHttpRequest
- Skrypty działające po stronie klienta
- Wymiana informacji z serwerem
- Biblioteka Microsoft Ajax Client Library
- Używanie rozszerzeń serwerowych ASP.NET Ajax
- Kontrolki
- Ajax Control Toolkit
- Korzystanie z usług sieciowych przy użyciu Ajaksa
- Lokalizacja i globalizacja w ASP.NET Ajax
- Debugowanie i śledzenie aplikacji ASP.NET
- Budowa przykładowej aplikacji handlu elektronicznego

Przyspieszony kurs praktycznego wykorzystania ASP.NET Ajax

Spis treści

O autorach	15
Podziękowania	17

Część I Podstawy technologii Ajax

Rozdział 1. Podstawy technologii ASP.NET Ajax	21
Co trzeba umieć	22
Ajax — zmiana modelu	22
Elementy technologii Ajax	23
Zalety i wady technologii Ajax	24
Wady technologii Ajax	24
Trochę historii	25
Co to jest ASP.NET Ajax	26
Inne frameworki Ajax	27
Funkcje ASP.NET Ajax	28
Instalowanie Ajaksa	28
Przygotowywanie środowiska	28
Instalowanie ASP.NET Ajax	29
Pierwsza aplikacja Ajax	31
Tworzenie ogólnej funkcji instalującej obiekt XMLHttpRequest	32
Jak to działa	33
Jak działa powyższy kod	35
Podsumowanie	37
Warsztat	38
Test	38
Odpowiedzi	38
Rozdział 2. Architektura ASP.NET Ajax	41
Wprowadzenie do ASP.NET	41
Architektura ASP.NET	42
Zdarzenia cyklu życia aplikacji	42

Zdarzenia cyklu życia strony	43
Architektura ASP.NET Ajax	46
Co znajduje się wewnątrz API serwerowego ASP.NET Ajax	48
Co znajduje się wewnątrz API klienckiego ASP.NET Ajax	50
Podsumowanie	51
Warsztat	51
Test	51
Odpowiedzi	51
Rozdział 3. Obiekt XMLHttpRequest	53
Krótki opis obiektu XMLHttpRequest	53
Trochę historii	54
Tworzenie obiektu XMLHttpRequest	54
Pobieranie danych synchronicznie i asynchronicznie przy użyciu obiektu XMLHttpRequest	56
Synchroniczne pozyskiwanie danych	56
Asynchroniczne pozyskiwanie danych	58
Praca z obiektem XMLHttpRequest	61
Symulacja Ajaksa bez obiektu XMLHttpRequest	66
Podsumowanie	67
Warsztat	67
Test	67
Odpowiedzi	68
Rozdział 4. Skrypty działające po stronie klienta	69
Wprowadzenie do DHTML	69
Co to jest CSS	70
Obsługa zdarzeń w języku JavaScript	75
JavaScript i model obiektów dokumentu	76
Obiekt Document	79
Obiekt Element	80
Implementacja skryptu działającego po stronie klienta	80
Podsumowanie	83
Warsztat	84
Test	84
Odpowiedzi	84

Rozdział 5. Wymiana informacji z serwerem	87
Cykl żądania i odpowiedzi	87
Formaty wymiany danych	89
Format HTML	89
Najbardziej popularny czysty tekst	89
XML — język internetu do wymiany danych	90
Wprowadzenie do JSON	91
Przechowywanie zbiorów uporządkowanych elementów w tablicach	93
Przechowywanie par nazwa-wartość w literałach obiektowych	94
Format JSON	95
Analiza danych w formacie JSON	96
Wykorzystanie formatu JSON w Ajaksie	97
Podsumowanie	100
Warsztat	101
Test	101
Odpowiedzi	101
Rozdział 6. Biblioteka Microsoft Ajax Client Library	103
Podstawowe wiadomości na temat biblioteki Microsoft Ajax Client Library	104
Funkcje biblioteki Microsoft Ajax Client Library	104
Przestrzenie nazw biblioteki Microsoft Ajax Client Library	105
Przestrzeń nazw Global — rozszerzanie JavaScriptu	106
Przestrzeń nazw Sys — podstawa wszystkich przestrzeni nazw	106
Przestrzeń nazw Sys.Net	107
Przestrzeń nazw Sys.Serialization	107
Przestrzeń nazw Sys.Services	107
Przestrzeń nazw Sys.UI	107
Przestrzeń nazw Sys.WebForms — częściowe wizualizowanie	108
Budowa biblioteki Microsoft Ajax Client Library	108
Budowa głównego systemu	109
System interfejsu użytkownika	111
Rozszerzanie biblioteki JavaScript za pomocą systemu rozszerzeń JavaScript Microsoft Ajax	113
Podsumowanie	115
Warsztat	115
Test	115
Odpowiedzi	116

Część II Ajax w praktyce

Rozdział 7. Używanie rozszerzeń serwerowych ASP.NET Ajax	121
System rozszerzeń serwerowych ASP.NET Ajax	122
Komponenty systemu rozszerzeń serwerowych ASP.NET Ajax	124
Kontrolki serwerowe ASP.NET Ajax	125
Biblioteka Microsoft Ajax Server Reference	128
Przestrzeń nazw System.Web.UI	128
Przestrzeń nazw System.Web.UI.Design	130
Przestrzeń nazw System.Web.Configuration	130
Przestrzeń nazw System.Web.Handlers	131
Przestrzeń nazw System.Web.Script.Serialization	131
Przestrzeń nazw System.Web.Script.Services	131
Podsumowanie	131
Warsztat	132
Test	132
Odpowiedzi	132
Rozdział 8. Używanie kontrolek UpdatePanel i UpdateProgress	135
Częściowe odświeżanie	135
Dlaczego częściowe odświeżanie jest warte zachodu	136
Trochę historii	137
Kontrolka serwerowa UpdatePanel	
— niezbędna w implementacji częściowego wizualizowania stron	138
Znacznik ContentTemplate	141
Co to są wyzwalacze	146
Kontrolka UpdateProgress	148
Podsumowanie	150
Warsztat	150
Test	150
Odpowiedzi	151
Rozdział 9. Używanie kontrolek ScriptManager i Timer	153
Kontrolka ScriptManager	153
Obsługa błędów przy użyciu Ajaksa	155
Kontrolka Timer	158
Implementacja funkcji częściowego odświeżania w aplikacjach	
ASP.NET Ajax przy użyciu kontrolek UpdatePanel i Timer	160
Podsumowanie	163

Warsztat	163
Test	163
Odpowiedzi	164
Rozdział 10. Ajax Control Toolkit — część I	165
Podstawowe informacje na temat ASP.NET Ajax Control Toolkit	165
Czym są kontrolki Ajax	166
Kontrolki rozszerzające	169
Kontrolka skryptowa	170
Biblioteka Control Toolkit	171
Kontrolka rozszerzająca AutoComplete	171
Podsumowanie	177
Warsztat	177
Test	177
Odpowiedzi	178
Rozdział 11. Ajax Control Toolkit — część II	179
Kontrolka rozszerzająca ConfirmButton	179
Kontrolka rozszerzająca DropDown	188
Podsumowanie	191
Warsztat	192
Test	192
Odpowiedzi	192
Rozdział 12. ASP.NET Ajax i formanty Web Part	193
Wprowadzenie do Web Parts	194
Właściwości formantów Web Part	195
Tworzenie formantów Web Part	198
Niestandardowe formanty Web Part	201
Tworzenie formantów Web Part przy użyciu kontrolki użytkownika	202
Ajax w formantach Web Part	205
Metoda z użyciem kontrolki UpdatePanel	206
Wywołania zwrotne po stronie klienta	207
Podsumowanie	212
Warsztat	212
Test	212
Odpowiedzi	213

Rozdział 13. Zdarzenia cyklu życia strony ASP.NET Ajax w kliencie	215
Model zdarzeń ASP.NET Ajax po stronie klienta	215
Obsługa wyjątku PageRequestManagerParserErrorException	219
Powodowanie wyjątku PageRequestManagerParserErrorException	221
Unikanie możliwości wystąpienia błędu PageRequestManagerParserErrorException	223
Podsumowanie	224
Warsztat	224
Test	224
Odpowiedzi	224

Część III Techniki zaawansowane

Rozdział 14. Korzystanie z usług sieciowych przy użyciu Ajaksa	229
Warstwa komunikacji asynchronicznej	229
Zadania warstwy komunikacji asynchronicznej	230
Wysyłanie żądania HTTP z klienta	231
Wywoływanie usług sieciowych z poziomu skryptów klienckich	234
Klasy proxy usług sieciowych	237
Klasy proxy metod stron	238
Podsumowanie	240
Warsztat	240
Test	240
Odpowiedzi	240
Rozdział 15. Korzystanie z usługi uwierzytelniania przy użyciu Ajaksa	243
Uwierzytelnianie i jego rodzaje	244
Korzystanie z usług uwierzytelniania z poziomu skryptów klienckich	245
Przykładowa implementacja aplikacji	246
Podsumowanie	254
Warsztat	254
Test	254
Odpowiedzi	255
Rozdział 16. Korzystanie z usługi profili użytkowników przy użyciu Ajaksa	257
Praca z usługą profili użytkownika	257
Włączanie usługi Profile	258
Definiowanie sekcji profilu	258

Implementacja przykładowej aplikacji	259
Podsumowanie	271
Warsztat	271
Test	271
Odpowiedzi	271
Rozdział 17. Rozszerzanie biblioteki Microsoft Ajax Library	273
Rozszerzanie biblioteki Microsoft Ajax Library	273
Rozszerzanie biblioteki Ajax za pomocą komponentów	274
Rozszerzanie biblioteki Ajax za pomocą kontrolek	279
Rozszerzanie biblioteki Ajax za pomocą zachowań	280
Podsumowanie	281
Warsztat	281
Test	281
Odpowiedzi	281
Rozdział 18. Lokalizacja i globalizacja w ASP.NET Ajax	283
Lokalizacja i globalizacja	283
Globalizacja i lokalizacja skryptowa przy użyciu JavaScriptu	288
Osadzanie zasobów skryptowych w asemblacjach	291
Używanie asemblacji z osadzonymi skryptami i zasobami	294
Podsumowanie	295
Warsztat	295
Test	295
Odpowiedzi	295
Rozdział 19. Debugowanie i śledzenie aplikacji ASP.NET	297
Debugowanie i śledzenie	298
Klasa Sys.Debug	298
Techniki debugowania kodu	302
Włączanie opcji debugowania w pliku web.config	304
Włączanie opcji debugowania w przeglądarce Internet Explorer	304
Włączanie opcji debugowania w Visual Studio	306
Podsumowanie	306
Warsztat	307
Test	307
Odpowiedzi	307

Rozdział 20. Pakiet ASP.NET Ajax Futures CTP	309
Pakiet ASP.NET Ajax Futures CTP	309
Czysto klienckie kontrolki w przestrzeni nazw Sys.Preview.UI	310
Dynamiczne witryny sterowane danymi	311
Dynamiczne kontrolki danych — przetwarzanie danych przy użyciu mniejszej ilości kodu	312
Zwracanie egzemplarzy obiektów DataSet, DataTable oraz DataRow	316
Przyszłość Ajaksa	317
Podsumowanie	318
Warsztat	318
Test	318
Odpowiedzi	319

Część IV Budowa przykładowej aplikacji handlu elektronicznego przy użyciu technologii ASP.NET Ajax

Rozdział 21. Podstawy handlu elektronicznego i projektowanie aplikacji	323
Podstawy handlu elektronicznego	324
Moduły aplikacji	324
Strona główna oraz formularz rejestracyjny i logowania	325
Wyświetlanie i wyszukiwarka produktów oraz koszyk na zakupy	325
Generowanie zamówień i płatności	326
Zarządzanie kontami użytkowników i rolami	326
Zarządzanie zamówieniami i produktami	327
Architektura i przepływ sterowania	327
Projekt bazy danych	328
Tabele i relacje	332
Podsumowanie	340
Warsztat	340
Test	340
Odpowiedzi	340
Rozdział 22. Początek pracy nad programem	343
Strona wzorcowa	343
Użytkownicy i role	349
Rejestrowanie klienta	353

Nawigacja	362
Podsumowanie	367
Warsztat	368
Test	368
Odpowiedzi	368
Rozdział 23. Wyszukiwanie i kupowanie produktów	371
Tworzenie kategorii, obrazów i produktów	371
Tworzenie obiektów biznesowych	374
Dodawanie i edytowanie kategorii	379
Wyszukiwarka produktów	388
Strona szczegółów produktu	393
Dodawanie produktów do koszyka	395
Zarządzanie koszykiem	396
Podsumowanie	402
Warsztat	402
Test	402
Odpowiedzi	403
Rozdział 24. Generowanie i obsługa zamówień	405
Składanie zamówienia	406
Szczegóły zamówienia	406
Podawanie adresu przesyłki	408
Szczegóły płatności	410
Stan zamówienia	414
Obsługa oczekujących zamówień	415
Historia zamówień	419
Podsumowanie	424
Warsztat	424
Test	424
Odpowiedzi	424
Skorowidz	427

Rozdział 5

Wymiana informacji z serwerem

W rozdziale:

- ▶ formaty wymiany danych między klientem a serwerem,
- ▶ wprowadzenie do JSON,
- ▶ używanie JSON.

W rozdziale 2. „Architektura ASP.NET Ajax” opisaliśmy architekturę ASP.NET Ajax. Dowiedziałeś się, z czego składają się biblioteka Microsoft Ajax Client Library oraz rozszerzenia serwera ASP.NET. Jedną z warstw biblioteki Client Library jest warstwa komponentów (*Component Layer*) składająca się z kilku niewidocznych komponentów umożliwiających asynchroniczną komunikację oraz obsługujących serializację XML i JSON. Serwer ASP.NET komunikuje się z biblioteką Client Library przez warstwę komponentów tej biblioteki. W tym rozdziale opiszemy JSON (wym. jak Jason) — ang. *JavaScript Object Notation* — oraz XML i formaty łańcuchowe.

Cykl żądania i odpowiedzi

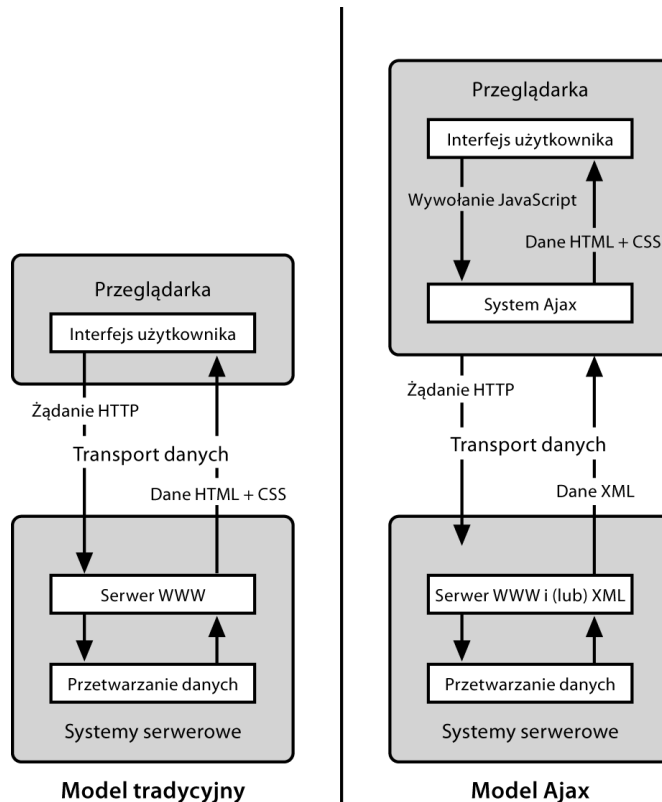
Zanim zaczniemy zgłębiać tematykę formatów wymiany danych w ASP.NET Ajax, sprawdzimy, czym różni się tradycyjny model aplikacji sieciowej od modelu aplikacji Ajax. W tym pierwszym zatwierdzenie formularza powoduje wysłanie żądania do serwera WWW, który wykonuje odpowiednie czynności i wysyła w odpowiedzi kompletną stronę internetową. Proces ten pochłania bardzo dużo transferu, ponieważ wymaga wysłania całej strony do serwera i zwrócenia przez niego w odpowiedzi takiej samej lub nowej. Bardzo często znaczna część kodu HTML wysyłanego przez serwer jest taka sama jak w żądaniu. Dzieje się tak

w każdym przypadku interakcji użytkownika z serwerem. W wyniku tego wydłuża się czas odpowiedzi serwera, co z kolei ma ujemny wpływ na wydajność całej aplikacji i komfort użytkownika.

Gdy aplikacja Ajax wyśle żądanie do serwera, ten w odpowiedzi odsyła tylko te dane, które są potrzebne. Do najczęściej wykorzystywanych formatów danych w tym przypadku należą: SOAP, HTML, XML, czysty tekst lub JSON. Przetwarzaniem tych danych w kliencie zajmuje się JavaScript. W tym modelu serwer z klientem wymieniają znacznie mniejsze ilości informacji, dzięki czemu aplikacja reaguje szybciej. Zwiększa się komfort użytkownika, ponieważ serwer WWW oszczędza mnóstwo czasu.

Na rysunku 5.1 przedstawiono porównanie tradycyjnego modelu aplikacji i opartego na Ajaksie

RYSUNEK 5.1.
Tradycyjny model aplikacji a aplikacja Ajax



Formaty wymiany danych

Jak napisaliśmy w poprzednim podrozdziale, w aplikacjach Ajax większy nacisk jest kładziony na dane niż treść przesyłaną w sieci, dlatego postanowiliśmy poświęcić nieco miejsca na opis dostępnych formatów wymiany danych. Wybór jednego z nich zależy od potrzeb danej aplikacji. Poniżej znajduje się lista kilku takich formatów:

- ▶ HTML,
- ▶ czysty tekst i format łańcuchowy,
- ▶ XML (*Extensible Markup Language*),
- ▶ JSON (*JavaScript Object Notation*).

Każdy z nich opiszemy w kolejnych podrozdziałach.

Format HTML

Jednym z najczęściej używanych formatów wymiany danych między serwerem a klientem jest HTML. Jeśli serwer wyśle odpowiedź na żądanie w tym formacie, dostęp do danych można uzyskać za pomocą JavaScriptu i wstawić je w dowolnym elemencie za pomocą własności `innerHTML` lub pokrewnych metod. W naszym przykładzie z listą elementów treść HTML wysyłana do przeglądarki byłaby następująca:

```
<div><span>HP</span><span>5446 A</span>  
<span>2007</span><span>$ 896.00</span></div>  
<div><span>Compaq</span><span>654AN</span>  
<span>2006</span><span>$ 655.00</span></div>  
<div><span>DELL </span><span>34543656</span>  
<span>2007</span><span>$ 720.00</span></div>
```

Treść tę można wstawić w tej postaci do każdego elementu, który ma własność `innerHTML`, łatwo jest ją zatem związać z wybranym elementem, formatowanie danych HTML odebranych z serwera może być jednak kłopotliwe.

Najbardziej popularny czysty tekst

Czysty tekst i format łańcuchowy to prosty format wymiany danych. Serwer zwraca sam tekst, który można związać z dowolnym elementem HTML za pomocą własności `value`, `text` lub `innerText`, w zależności od elementu. Można też użyć oddzielanych wartości łańcuchowych, takich jak format odpowiedzi

spokrewnionych danych, chociaż opcja ta ma pewne wady. Niektóre kontrolki serwerowe ASP.NET Ajax, o których piszemy dalej w książce, wysyłają w odpowiedzi oddzielane wartości łańcuchowe. Jedną z nich jest `UpdatePanel`, która wysyła odpowiedzi w formacie:

```
Size|ControlType|ControlName|ControlData
```

Znakiem oddzielającym w tym przykładzie jest pionowa kreska `|`. `Size` oznacza liczbę bajtów, `ControlType` to rodzaj kontrolki (w tym przypadku `UpdatePanel`), a `ControlName` to jej nazwa. Ostatnia wartość `ControlData` zawiera dane HTML, które mają zostać wyświetlone w przeglądarce. Zaletą tego formatu jest prostota. Wadą natomiast to, że jeśli zmieni się kolejność wartości, konieczne jest modyfikowanie kodu działającego po stronie klienta.

XML — język internetu do wymiany danych

XML to język znaczników przeznaczony do opisu danych. Skrót XML pochodzi od angielskich słów *Extensible Markup Language* (rozszerzalny język znaczników). Opracowywaniem jego standardów zajmuje się organizacja W3C, która opublikowała jego rekomendację. Obecnie XML jest popularnym formatem wymiany danych i manipulacji nimi. Jest jednym z najczęściej używanych formatów w aplikacjach internetowych i okienkowych, ponieważ został przyjęty przez wielu producentów.

Wymiana danych w formacie XML między serwerem a klientem jest popularnym rozwiązaniem, ponieważ większość współczesnych przeglądarek posiada implementację DOM XML, co ułatwia przetwarzanie z poziomu JavaScriptu. XML nie posiada żadnych standardowo zdefiniowanych znaczników, wszystkie więc należy zdefiniować samemu. Do opisu danych wykorzystuje się definicję typu dokumentu (ang. *Document Type Definition* — DTD) lub XML Schema. Innymi słowy, DTD lub XML Schema sprawdzają gramatykę, aby zapewnić poprawność danych. Dzięki prostocie reprezentacji danych i możliwości przedstawiania ich w postaci hierarchicznej podejście to zyskało sobie bardzo dużą popularność. Ponadto XML jest dostępny bezpłatnie i można go rozszerzać.

Poniżej znajduje się przykładowa reprezentacja danych w formacie XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ItemList>
  <Item>
    <Name>HP</Name>
    <Model>5446 A</Model>
```

```
<Make>2007</Make>
<Price>$ 896.00</Price>
</Item>
<Item>
  <Name>Compaq</Name>
  <Model>654AN</Model>
  <Make>2006</Make>
  <Price>$ 655.00</Price>
</Item>
<Item>
  <Name>DELL</Name>
  <Model>34543656</Model>
  <Make>2007</Make>
  <Price>$ 720.00</Price>
</Item>
</ItemList>
```

Powyzsze dane XML reprezentują elementy `Item` i ich atrybuty: `Name`, `Model`, `Make` oraz `Price`. Ta prosta struktura mówi sama za siebie.

Dane XML pobrane z serwera można wyświetlić w przeglądarce na kilka sposobów. Są to:

- ▶ **Użycie specyfikacji DOM XML** — za pomocą tego API można przeprowadzić analizę danych XML, uzyskać do nich dostęp oraz przekształcić je na HTML za pomocą JavaScriptu. Więcej informacji dotyczących specyfikacji API DOM XML znajdziesz na stronie www.w3.org.
- ▶ **Transformacja dokumentu na HTML za pomocą XSLT** — XSLT to oparty na XML-u deklaratywny język transformujący, za pomocą którego można przekształcić dokument XML w jednym formacie na inny. Procesor XSLT przy użyciu arkusza stylów XSLT przekształca dokument XML na HTML, czysty tekst lub jakiś inny format obsługiwany przez niego. Język XSLT to zagadnienie, o którym można napisać osobną książkę. Polecamy zatem *Sams Teach Yourself XSLT In 21 Days*, której autorem jest Michiel van Otegem.

Wprowadzenie do JSON

Gdy około 2004 roku zaczęły pojawiać się usługi sieciowe oparte na XML-u, język ten stał się standardowym formatem do przesyłania informacji. W aplikacjach Ajax do danych XML można uzyskać dostęp poprzez obiekt `XMLHttpRequest`. Własność `responseXML` umożliwia dostęp w formacie XML, a `responseText` — w formacie łańcuchowym.

XML jako format wymiany danych ma jednak pewne wady. Źle sprawdza się, gdy między serwerem a klientem trzeba przesłać dużą ilość informacji. Dlaczego? Po pierwsze, analiza i dostęp do wielkich ilości danych XML są trudne. Po drugie, dane w formacie XML zawierają więcej bajtów, niż gdyby zastosowano jakiś oszczędniejszy format.

Douglas Crockford opracował alternatywę — niezwykle lekki format oparty na łańcuchach, za pomocą którego można wymieniać dane między serwerem a klientem. Jego nazwa to *JavaScript Object Notation* (JSON).

Wskazówka

Format JSON jest wykorzystywany do wymiany danych między Microsoft Ajax Library a ASP.NET Ajax. Z udziałem JSON odbywa się większa część operacji wymiany danych — około 80%. Reszta odbywa się przy użyciu XML-a i formatów łańcuchowych.

JSON obsługuje dwie struktury danych: obiekty i tablice. Obiekty, wyznaczone przez znaki `{ i }`, to nieuporządkowane zbiory par nazwa-wartość. Nazwy są oddzielone od wartości znakiem `:`, a pary nazwa-wartość — znakiem `,`.

Tablice to uporządkowane szeregi wartości. Ograniczają je znaki `[i]`, a wartości są oddzielane znakami `,`.

Nazwa to łańcuch w podwójnym cudzysłowie. Wartości mogą być jednego z następujących typów: `String`, `Number`, `Boolean` (`true` lub `false`), `Object`, `Array` oraz `null`.

Format JSON jest elastyczny, to znaczy umożliwia reprezentowanie dowolnej struktury danych w stanie, w jakim jest, oraz pozwala na dodawanie nowych pól, nie zakłócając pracy istniejących programów. Wielką zaletą tego formatu jest to, że jest bardziej zwięzły i łatwiejszy do zanalizowania niż XML. Programista musi tylko przekazać łańcuch w tym formacie do funkcji JavaScript `eval()`. Ta przetwarza przekazany łańcuch i wynik swojego działania zapisuje w elementach HTML. Należy jednak podkreślić, że użycie funkcji `eval()` niekorzystnie odbija się na wydajności. Można rozważyć użycie innej opcji, jeśli ilość danych jest naprawdę wielka — konwertowanie ogromnych łańcuchów na kliencie może spowodować opóźnienia. Najlepsze jednak jest to, że przetwarzanie można wykonać w pliku `.js` dołączonym do dowolnych stron. Ponadto JSON nie ma numeru wersji, ponieważ specyfikacja została określona jako stabilna na zawsze. Istnieje kilka parserów JSON, które obsługują większość języków programowania. Można je pobrać w witrynie www.json.org.

Większość kontrolki serwerowych ASP.NET Ajax wymienia się danymi w formacie JSON. Kontrolka wysyła łańcuch JSON, a biblioteka Microsoft Ajax Library przetwarza go przy użyciu wewnętrznych plików *.js* i przekazuje wynik do klienta. Na format ten zdecydowano się ze względu na rozmiar, zwięzłość oraz łatwość przetwarzania. Pozwala on polepszyć szybkość działania aplikacji ASP.NET Ajax. Przed zapoznaniem się z formatem JSON należy dowiedzieć się nieco na temat literałów tablicowych i obiektów w JavaScriptcie.

Przechowywanie zbiorów uporządkowanych elementów w tablicach

Czym są tablice? Są to uporządkowane szeregi wartości. Granice tablicy wyznaczają znaki `[]`. Można je stworzyć za pomocą konstruktora lub wpisać wartości bezpośrednio w nawiasie. Poniżej znajduje się deklaracja tablicy przy użyciu konstruktora:

```
var myItems = new Array();
```

Elementy do tej tablicy można dodać przy użyciu nawiasów i wartości wyznaczającej indeks, który określa położenie elementu w tablicy:

```
myItems[0] = "HP";  
myItems[1] = "Compaq";  
myItems[2] = "DELL";
```

Ten sam obiekt można utworzyć w bardziej wydajny sposób, posługując się literałem tablicowym:

```
var myItems = ["HP", "Compaq", "DELL"];
```

Jeszcze jeden sposób zadeklarowania opisywanej tablicy:

```
var myItems = new Array("HP", "Compaq", "DELL");
```

W języku JavaScript typ tablic nie jest kontrolowany. Dzięki temu można w nich przechowywać dane różnych typów.

Uwaga
Uwaga

Mimo że tablice można tworzyć za pomocą konstruktora, JSON akceptuje tylko sposób literałowy:

```
JSON format: ["HP", "Compaq", "DELL"]
```

Przechowywanie par nazwa-wartość w literałach obiektowych

Zadaniem obiektów jest przechowywanie par nazwa-wartość. Za ich pomocą można przechowywać dane biznesowe w formie obiektów. Pary są otoczone klamrami { i }. Nazwa i wartość w każdej parze są oddzielone znakiem :, a poszczególne pary oddziela się od siebie znakiem ,. Poniżej znajduje się przykład obiektu:

```
var objItem = {  
    "Name" : "HP",  
    "Model" : "5446 A",  
    "Make" : "2007",  
    "Price" : "$ 896.00"  
};
```

Powyższy kod tworzy obiekt objItem z atrybutami Name, Model, Make i Price, które mają przypisane wartości. Dostęp do każdego z nich można uzyskać przy użyciu nazwy obiektu i notacji z kropką:

```
objItem.Name // Dostęp do własności Name.  
objItem.Model // Dostęp do własności Model.  
objItem.Make // Dostęp do własności Make.  
objItem.Price // Dostęp do własności Price.
```

Alternatywny sposób dostępu do tych samych własności:

```
objItem["Name"];  
objItem["Model"];  
objItem["Make"];  
objItem["Price"];
```

Inne sposoby

Obiekt można także utworzyć za pomocą konstruktora przy użyciu słowa kluczowego new:

```
var objItem = new Object();
```

Teraz można dodać parametry i przypisać im wartości w następujący sposób:

```
objItem.Name = "HP";  
objItem.Model = "5446 A";  
objItem.Make = "2007";  
objItem.Price = "$ 896.00";
```

Własności można także dodać przy użyciu składni tablicowej:

```
objItem["Name"] = "HP";  
objItem["Model"] = "5446 A";  
objItem["Make"] = "2007";  
objItem["Price"] = "$ 896.00";
```

W JSON obiekt jest reprezentowany następująco:

```
{  
  "Name": "HP",  
  "Model": "5446 A",  
  "Make": "2007",  
  "Price": "$ 896.00"  
}
```

W JSON nie można używać konstruktorów do tworzenia tablic i obiektów. Służą do tego literały.

Uwaga
Uwaga

Format JSON

Składnia JSON jest zbiorem literałów tablicowych i obiektowych. W formacie tym nie ma jednak zmiennych, przypisań ani operatorów. Jest to tylko sposób reprezentacji danych i sam w sobie nie jest językiem. Jest to podzbiór JavaScriptu opisujący dane przy użyciu literałów tablicowych i obiektowych.

Jak pisaliśmy wcześniej, reprezentacja obiektu `Item` w JSON może wyglądać następująco:

```
{  
  "Name": "HP",  
  "Model": "5446 A",  
  "Make": "2007",  
  "Price": "$ 896.00"  
}
```

W typowym procesie komunikacji serwera z przeglądarką dane w formacie JSON są zwracane do przeglądarki jako łańcuch. Aby można go było użyć, musi zostać przekonwertowany na obiekt. Do tego służy funkcja `eval()`. Analizuje ona łańcuch i konwertuje go na obiekt JavaScript.

Załóżmy, że odbieramy dane w zmiennej `vItem`. Poniższa instrukcja konwertuje ten łańcuch JSON na obiekt:

```
var vItem = eval("(" + vItem + ")");
```

Teraz zmienna `vItem` zawiera obiekt `Item` pobrany z serwera w formacie JSON. Wartości można z niego pobrać w następujący sposób:

```
alert(vItem.Name); // wynik: "HP"  
alert(vItem.Model); // wynik: "5446 A"  
alert(vItem.Make); // wynik: "2007"  
alert(vItem.Price); // wynik: "$ 896.00"
```

Uwaga

Przy konwersji łańcucha JSON na obiekt za pomocą funkcji `eval()` konieczne jest zastosowanie dodatkowych nawiasów. Jest to spowodowane tym, że klamry mogą w JavaScriptcie zostać zinterpretowane przez instrukcję `if`, `for` lub jakąś inną konstrukcję językową.

Analiza danych w formacie JSON

Funkcja `eval()` jest ogólną funkcją JavaScript służącą do wykonywania lub analizowania różnych typów danych. Jeśli chcesz użyć specjalnego parsera JSON do tworzenia obiektów i tablic z tekstu JSON i odwrotnie, możesz użyć dostępnego pod adresem www.json.org/json.js. Plik ten można skopiować i dołączyć do swojej strony w nagłówku za pomocą poniższego wiersza kodu:

```
<script type="text/javascript" src="json.js"></script>
```

Znajdujący się w pliku `json.js` parser ma dwie podstawowe funkcje:

```
ParseJSON()  
toJSONString()
```

Pierwsza konwertuje tekst JSON na obiekt JavaScript, a druga – obiekty JavaScript na tekst lub łańcuch JSON.

Po dodaniu pliku `json.js` do strony funkcja `toJSONString()` zostaje dodana do definicji obiektów i tablic JavaScript. Poniżej znajduje się przykład użycia tej metody:

```
<script language="javascript">  
var myItem = new Object();  
myItem.Name = "HP";  
myItem.Model = "5446 A";  
myItem.Make = "2007";  
myItem.Year = "$ 896.00";  
myItem = myItem.toJSONString();  
alert("Reprezentacja obiektu Item jako łańcucha JSON: " + myItem);  
</script>
```

Skrypt ten zwraca następujący wynik:

Reprezentacja obiektu Item jako łańcucha

```
JSON: {"Name": "HP", "Model": "5446 A", "Make": "2007", "Price": "$ 896.00"}
```

Zauważ, że jeśli do wykonania operacji Ajax w ASP.NET użyjesz kontrolek serwerowych ASP.NET, takie rzeczy, jak analiza składni lub konwersja, są wykonywane wewnętrznie przez Microsoft Ajax Client Library.

Wykorzystanie formatu JSON w Ajaksie

W rozdziale 3. „Obiekt XMLHttpRequest” zademonstrowaliśmy pobieranie danych XML i łańcuchowych za pomocą obiektu XMLHttpRequest. Teraz zobaczymy, jak to się dzieje w przypadku danych w formacie JSON. Ponieważ jest to format tekstowy, dane te musimy pobierać przy użyciu własności `responseText` obiektu XMLHttpRequest. Jej implementację już opisaliśmy. Jedyna różnica będzie polegać na tym, że teraz będzie ona przechowywała dane w formacie JSON.

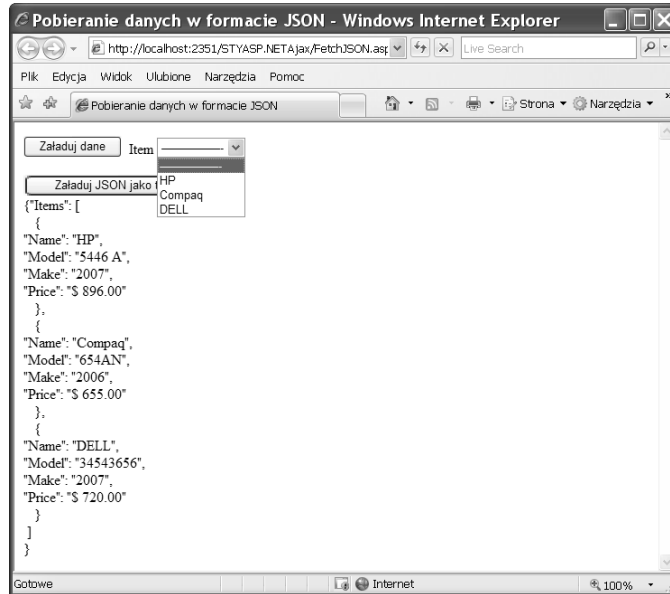
Posłużymy się tym samym programem, co w rozdziale 3., ale zamiast XML pobierzemy plik w formacie JSON. Aplikacja ta ma dwa przyciski: *Załaduj dane* oraz *Załaduj JSON jako tekst*. Po wczytaniu strony w przeglądarce lista rozwijana dla danych jest początkowo pusta. Kliknięcie przycisku *Załaduj dane* powoduje pobranie nazw elementów (które są w formacie JSON) ze znajdującego się na serwerze pliku tekstowego *Items.txt* i asynchroniczne wstawienie ich do listy. Kliknięcie *Załaduj JSON jako tekst* wstawia dane łańcuchowe z pliku *Items.txt* do znajdującego się na stronie elementu `div`. Dalej opisujemy wykonanie tego programu krok po kroku oraz przedstawiamy wynik jego działania w przeglądarce na rysunku 5.2.

Otwórz rozwiązanie, które utworzyłeś w rozdziale 3., i wykonaj poniższe czynności:

1. Dodaj nowy element *Web Form* i nazwij go *FetchJSON.aspx*.
2. Utwórz plik tekstowy o nazwie *Items.txt* i wprowadź do niego poniższe dane:

```
{"Items": [  
  {  
    "Name": "HP",  
    "Model": "5446 A",
```

RYSUNEK 5.2.
Strona
FetchJSON.aspx
w oknie
przeglądarki



```

"Make": "2007",
"Price": "$ 896.00"
  },
  {
    "Name": "Compaq",
    "Model": "654AN",
    "Make": "2006",
    "Price": "$ 655.00"
  },
  {
    "Name": "DELL",
    "Model": "34543656",
    "Make": "2007",
    "Price": "$ 720.00"
  }
]
}

```

3. Dodaj plik *Items.txt* do rozwiązania.

4. Otwórz stronę *FetchJSON.aspx* i wstaw poniższy kod:

```

<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="FetchJSON.aspx.cs"
Inherits="FetchJSON" %>
<!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 runat="server">

```

```
<title>Pobieranie danych w formacie JSON</title>
<script language="javascript" type="text/javascript"
src="xmlhttp.js"></script>
<script language="javascript" type="text/javascript">
  var xmlhttp = false;
  function LoadItemNames() {
    getXmlHttpRequestObject();
    xmlhttp.open("GET", "Items.txt", true);
    xmlhttp.onreadystatechange = function()
    {
      if (xmlhttp.readyState == 4) {
        // Jeśli kod stanu HTTP to 200, tzn. jeśli żądanie jest w porządku.
        if (xmlhttp.status == 200) {
          var sJson = xmlhttp.responseText;
          sJson = eval("(" + sJson + ")");
          // Referencja do znacznika <select> – ddlItems.
          var ddlItems = document.getElementById("ddlItems");
          for(var i = 0; i < sJson.Items.length; i++) {
            // Tworzy nowy element <option>.
            var newOption =
              document.createElement('option');
            // Przypisuje wartość i tekst do nowego znacznika.
            newOption.value = sJson.Items[i].Name;
            newOption.text = sJson.Items[i].Name;
            // Dodaje nowy element w znaczniku <select> ddlItems.
            ddlItems.options.add(newOption);
          }
        }
      }
    }
    xmlhttp.send(null);
  }

  function LoadItemText() {
    getXmlHttpRequestObject();
    xmlhttp.open("GET", "Items.txt", true);
    xmlhttp.onreadystatechange = function()
    {
      if (xmlhttp.readyState == 4) {
        // Jeśli kod stanu HTTP to 200, tzn. jeśli żądanie jest w porządku.
        if (xmlhttp.status == 200) {
          document.getElementById("lblText").innerHTML =
            xmlhttp.responseText;
        }
      }
    }
    xmlhttp.send(null);
  }
</script>
</head>
```

```

<body>
  <form id="form1" runat="server">
    <input type="button" id="btnLoadItemNames"
value="Załaduj dane"
onclick="LoadItemNames();" />&nbsp;
<span id="spnItemNames">Item</span>
    <select id="ddlItems">
      <option value="">-----</option>
    </select><br /><br />
    <input type="button" id="btnLoadItemText"
value="Załaduj JSON jako tekst"
onclick="LoadItemText();" />&nbsp;
    <div id="lblText"></div>
  </form>
</body>
</html>

```

5. Teraz możesz uruchomić aplikację, aby zobaczyć rezultat jej działania. Ustaw plik *FetchJSON.aspx* jako stronę startową i uruchom program, naciskając klawisz *F5*. Jeśli w pliku *web.config* wyłączone jest narzędzie debugowania, zostaniesz poproszony o jego włączenie. Po tym aplikacja zostanie otwarta w przeglądarce, jak widać na rysunku 5.2.

Jedyna zmiana w stosunku do poprzedniej wersji polega na użyciu własności `responseText` (która przechowuje dane w formacie JSON) w zmiennej zamiast obiektu `XML documentElement` (który był używany w rozdziale 3.). Teraz zmienna `sJson` zawiera dane z pliku *Items.txt* w postaci łańcucha i jest przepuszczana przez funkcję JavaScript `eval()`, która konwertuje je na obiekt JavaScript.

```

var sJson = xmlHttp.responseText;
sJson = eval("(" + sJson + ")");

```

Dane z obiektu `sJson` można pobrać przez kolekcję `Items`, jak demonstruje poniższy kod:

```

sJson.Items[i].Name; // Pobiera wartość własności Name.
// i jest indeksem w pętli.

```

Podsumowanie

W tym rozdziale zwięźle opisaliśmy różne formaty wymiany danych używane w cyklach żądanie-odpowiedź strony. Do najpopularniejszych należą: XML, czysty tekst i format łańcuchowy, HTML oraz JSON, który jest najnowszy i zdobywa

coraz większą popularność w aplikacjach sieciowych. W porównaniu z formatem XML JSON jest lżejszy, bardziej zwięzły oraz łatwiejszy do opanowania i konserwacji. Przenosi mniejszą liczbę bajtów niż XML, dzięki czemu aplikacje działają szybciej i lepiej reagują na działania użytkownika.

Warsztat

Test

1. Jak odbywa się analiza składni dokumentu XML?
2. Czym jest JSON?
3. Formatu JSON można używać w aplikacjach okienkowych. Prawda czy fałsz?
4. Jak odbywa się analiza składni danych w formacie JSON w przeglądarce?
5. Jak używa się formatu JSON w połączeniu z Ajaxem?
6. Wymień znane Ci formaty wymiany danych.
7. Jakie typy danych są obsługiwane przez format JSON?

Odpowiedzi

1. Dokument XML musi mieć poprawną strukturę. Aby przeprowadzić jego walidację, musi zostać dołączony do niego schemat w postaci DTD (*Document Type Definition*) lub XSD (*XML Schema Definition*).
2. JSON (ang. *JavaScript Object Notation*) to lekki format wymiany danych będący podzbiorem języka JavaScript. Obsługuje dwie struktury danych tego języka: literały obiektowe i tablicowe.
3. Fałsz. Ponieważ JSON jest formatem pochodnym od języka JavaScript, jest on analizowany przez system JavaScript w przeglądarce, dlatego nie jest formatem dla aplikacji okienkowych.
4. Dane JSON są odbierane w postaci łańcucha od obiektu odpowiedzi. Przetwarza je funkcja JavaScript `eval()`. Funkcja ta konwertuje łańcuch JSON na obiekt JavaScript, który można przeglądać.
5. Łańcuch JSON można pobrać w Ajaxie za pomocą obiektu `XMLHttpRequest`. Dane znajdują się we własności `responseText` tego obiektu.

6. Lista dostępnych formatów wymiany danych:

- ▶ HTML,
- ▶ czysty tekst i format łańcuchowy,
- ▶ XML,
- ▶ JSON.

Są to najpopularniejsze formaty, ale istnieją jeszcze inne.

7. JSON obsługuje następujące typy danych:

- ▶ String,
- ▶ Number,
- ▶ Boolean,
- ▶ Array,
- ▶ Object,
- ▶ Null.