

# KAPITOLA 6

## XML, XPath a XSLT

S rostoucí popularitou XML chtěli vývojáři tuto technologii použít na obou stranách – jak na straně serveru, tak i na straně uživatele. Microsoft a Mozilla, počínaje Internet Explorerem 5.0 a Mozillou 1.0 (předchůdcem Firefoxu), do svých prohlížečů implementovali podporu XML v JavaScriptu. Opera 8 a Safari 1.2 umožnily základní formu podpory XML. Zatímco Opera podporu XML v JavaScriptu neustále rozvíjí, Safari o něco zaostává a dnes má nejmenší podporu XML. Tvůrci prohlížečů pokračují v rozšiřování podpory XML pomocí různých nových rysů, čímž vývojářům nabízejí silné nástroje, které jsou podobné těm, jež bylo původně možné nalézt pouze na serveru.

## Podpora XML v prohlížečích

V současné době je dostupných mnoho webových prohlížečů, ale jen několik z nich má úplnou podporu XML a s tím

spojených technologií. Vedoucí postavení mezi nimi zaujímá Internet Explorer (IE) a Mozilla Firefox, těsně za nimi je Opera (verze 9). O velký kus dále za nimi se vleče Safari od Apple, který podporuje pouze základní rysy XML. Navzdory těmto rozdílům všechny zmíněné prohlížeče obsahují základní funkčnosti XML, takže v této části se můžeme zabývat těmito čtyřmi hlavními prohlížeči.

## XML DOM v IE

Microsoft přidal podporu XML do IE 5.0 tím, že do něj zařadil ActiveX knihovnu MSXML, což byla komponenta, která byla původně vytvořena pro zpracování aktivních kanálů v IE 4.0. Tato původní verze komponenty nebyla zamýšlena pro užívání veřejností, nicméně vývojáři ji objevili a začali ji používat. Microsoft pak zareagoval celkovou aktualizací verze MSXML, která byla obsažena v IE 4.01. Knihovna MSXML byla zpočátku součástí pouze IE.

Toto trvalo až do roku 2001, kdy Microsoft vydal MSXML 3.0 – samostatné rozšíření, které bylo dostupné skrze webové stránky společnosti. Později v tomto roce byla vydána verze 4.0 a knihovna MSXML se přejmenovala na Microsoft XML Core Services Component. Od svého vzniku MSXML prošla vývojem od základního a neplatného (nevalidního) XML parseru až po silnou komponentu, která může ověřovat platnost XML dokumentů, provádět XSL transformace, podporovat jmenné prostory, jednoduché API pro XML (SAX) či různé W3C standardy jako XPath nebo schémata XML. A každá nová verze je výkonnější.

## Vytvoření objektu XML DOM

Za účelem usnadnění vytváření objektů ActiveX v JavaScriptu zavedl Microsoft třídu zvanou `ActiveXObject`. Její konstruktor přijímá jeden argument – řetězec obsahující jméno a verzi objektu ActiveX, který má být vytvořen. V tomto případě se jedná o verzi XML dokumentu. První ActiveX objekt XML DOM byl pojmenován jako `Microsoft.XmlDom` a jeho vytvoření vypadá takto:

```
var oXmlDom = new ActiveXObject("Microsoft.XmlDom");
```

Nově vytvořený objekt XML DOM se nechová stejně jako každý jiný objekt DOM – umožňuje vám totiž procházet stromovou strukturou DOM a manipulovat s uzly DOM.

V době, kdy vznikala tato kniha, existovalo celkem 6 různých verzí MSXML DOM. Jedná se o následující řetězce verzí:

- `Microsoft.XmlDom`.
- `MSXML2.DOMDocument`.
- `MSXML2.DOMDocument.3.0`.
- `MSXML2.DOMDocument.4.0`.
- `MSXML2.DOMDocument.5.0`.
- `MSXML2.DOMDocument.6.0`.

*Knihovna MSXML je dostupná pouze na Internet Exploreru ve Windows. Internet Explorer 5 na počítačích Mac nemá podporu XML DOM.*

Protože každá nově vydaná knihovna MSXML přináší mnoho vylepšení, měli byste vždy používat tu nejnovější. Microsoft doporučuje kontrolovat existenci nejnovějších verzí (v době psaní této knihy to byla MSXML6). Dále doporučuje v případě problémů používat verzi MSXML3. Z pohledu vývojáře tedy bude užitečné vytvořit funkci, jež by určila, kterou verzi použít. Následující funkce `createDocument()` vytvoří MSXML6 DOM, pokud ji počítač uživatele podporuje. V opačném případě je vytvořena MSXML3 DOM:

```
function createDocument() {  
    var aVersions = [  

```

```
"MSXML2.DOMDocument.6.0",
"MSXML2.DOMDocument.3.0",
];
for (var i = 0; i < aVersions.length; i++) {
    try {
        var oXmlDom = new ActiveXObject(aVersions[i]);
        return oXmlDom;
    } catch (oError) {
        //Nedělej nic
    }
}
throw new Error("MSXML is not installed.");
}
```

Tato funkce provádí iterace skrz pole `aVersions`, jež zahrnuje řetězce verzí. Začíná s nejnovější verzí – `MSXML2.DOMDocument.6.0` – a snaží se vytvořit dokument DOM. Pokud je vytvoření objektu úspěšné, je vrácen a funkce `createDocument()` je ukončena. V opačném případě je vyhozena výjimka, která je pak zachycena pomocí bloku `try...catch`, takže smyčka pokračuje a vyzkouší se další verze. Pokud tvorba MSXML DOM dokumentu selže po dvou pokusech, je vyhozena výjimka, že knihovna MSXML není nainstalována. Volání funkce vypadá takto:

```
var oXmlDom = createDocument();
```

V tuto chvíli máte XML dokument k dispozici a je čas načíst nějaká data XML.

## Načítání XML dat v IE

Dokument MSXML DOM podporuje dvě metody načítání dat XML – `load()` a `loadXML()`. Metoda `load()` přijímá jeden argument, což je URL, ze které má být stažen XML soubor. Metoda `loadXML()` také přijímá jeden argument, jedná o řetězec dat XML. Obě metody mají za následek analýzu (parsování) dat XML a vytvoření struktury XML DOM.

Metoda `load()` se chová podobně jako XHR v tom, že může načítat data z externího souboru ve dvou režimech: asynchronním nebo synchronním. To nastavíte ve vlastnosti `async`. Standardně má `async` hodnotu `true`, takže `load()` metoda je asynchronní. Pro použití synchronního režimu musí být vlastnost `async` nastavena na `false`:

```
oXmlDom.async = false;
```

*Obecně je považováno za nepraktické vykonávat požadavky v synchronním režimu (kvůli možnosti zamrznutí uživatelského rozhraní). Synchronní režim by měl být používán šetrně a pouze v případě, kdy je ze serveru posíláno velmi malé množství dat.*

V asynchronním režimu vystavuje objekt MSXML vlastnost `readyState`, která má v podstatě pět stejných stavů jako XHR vlastnost `readyState` (popsáno v kapitole 2). Výjimkou je, že objekt MSXML nemá stav 0 (UNINITIALIZED). Dokument DOM navíc podporuje ovladač události `onreadystatechange`, který umožňuje sledovat vlastnost `readyState`:

```
oXmlDom.onreadystatechange = function () {  
    if (oXmlDom.readyState == 4) {  
        //Dělej něco, když jsou data kompletně načtena.  
    }  
};  
oXmlDom.load("myxml.xml");
```

V tomto příkladě je do XML DOM načten fiktivní XML dokument s názvem `myxml.xml`. Když `readyState` dosáhne hodnoty 4, je dokument plně načten a kód uvnitř bloku `if` bude proveden.

*Všimněte si, že na rozdíl od objektu XHR neexistuje v objektu XML DOM vlastnost `status`.*

Druhá možnost, jak načíst XML data – metoda `loadXML()` – je poněkud jednodušší a nevyžaduje žádné HTTP příkazy, protože data jsou již přítomna v počítači klienta. Předaná data musí obsahovat správně zformované XML, jako v následujícím příkladu:

```
var sXml = "<root><person><name>Jeremy McPeak</name></person></root>";  
oXmlDom.loadXML(sXml);
```

V tomto případě jsou XML data obsažena v proměnné `sXML` a jsou načítána do dokumentu `oXmlDom`. Není zde žádný důvod kontrolovat vlastnost `readyState` nebo nastavovat vlastnost `async`, protože neobsahuje příkazy serveru – data jsou načítána synchronně a jsou dostupná okamžitě.

### Kontrola platnosti XML dat v průběhu načítání

Objekt MSXML DOM standardně ověřuje platnost XML dokumentu, když analyzuje data. Platný XML dokument je takový dokument, který obsahuje referenci na definici typu dokumentu (DTD) v deklaraci `DOCTYPE` a přizpůsobuje se tomuto DTD.

Může se stát, že toto chování nebude žádoucí. V takovém případě bude vhodnější, aby byla zkontrolována pouze struktura dokumentu. Aby to bylo možné, poskytuje objekt MSXML DOM vlastnost `validateOnParse`. Povolené hodnoty jsou `true` (výchozí), nebo `false`, přičemž by měla být nastavena před tím, než objekt DOM načte dokument.

```
var oXmlDom = createDocument();  
oXmlDom.async = false;  
oXmlDom.validateOnParse = false;  
oXmlDom.load("myxml.xml");
```

V tomto kódu, kdy objekt XML DOM načítá a analyzuje kód XML, bude kontrola probíhat pouze za účelem kontroly správné struktury dokumentu.

### Ochrana prázdných znaků

MSXML DOM zachází s prázdnými znaky (whitespace) jinak než je standardem v DOM. MSXML DOM standardně odstraňuje z dokumentu pouze uzly s prázdnými znaky – nenechává nic kromě XML a textových uzlů. Zatímco mnozí považují tuto vlastnost za rozumnou, skutečnost je taková, že je to docela nepraktické.

MSXML DOM ovšem nabízí vlastnost `preserveWhiteSpace`, která říká parseru, aby uzly s prázdnými znaky zamítl nebo povolil. Tato vlastnost má logickou hodnotu – výchozí je `false`. Následující kód načítá XML dokument a zabráňuje odstranění prázdných znaků v něm:

```
var oXmlDom = createDocument();
oXmlDom.async = false;
oXmlDom.preserveWhiteSpace = true;
oXmlDom.load("myxml.xml");
```

Pokud je tato vlastnost `preserveWhiteSpace` nastavena na `true`, dovoluje, aby se objekt MSXML DOM choval jako standardní DOM.

### Procházení XML DOM v IE

Navigace v dokumentu XML DOM je podobná navigaci dokumentu DOM HTML – je to struktura hierarchicky uspořádaných uzlů. Na vrcholu stromu je `documentElement`, který obsahuje kořenový prvek dokumentu. Z tohoto místa můžete zpřístupnit kterýkoliv prvek nebo atribut dokumentu pomocí vlastností uvedených v tabulce 6-1.

**Tabulka 6-1. Vlastnosti XML DOM.**

Vlastnost	Popis
<code>attributes</code>	Kolekce atributů pro tento uzel.
<code>childNodes</code>	Kolekce potomků (dceřiných uzlů).
<code>firstChild</code>	První potomek uzlu.
<code>lastChild</code>	Poslední potomek uzlu.
<code>nextSibling</code>	Uzel bezprostředně následující po aktuálním uzlu.
<code>nodeName</code>	Jméno uzlu.
<code>nodeType</code>	XML DOM typ uzlu.
<code>nodeValue</code>	Text spojený s uzlem, pokud existuje.
<code>ownerDocument</code>	XML DOM dokument, jehož částí uzel je.

Vlastnost	Popis
ParentNode	Rodičovský uzel aktuálního uzlu.
PreviousSibling	Uzel bezprostředně předcházející aktuálnímu uzlu.
Text	Vrací obsah uzlu nebo zřetěžený text současného uzlu a jeho potomků.
Xml	Vrací XML řetězec reprezentující současný uzel a jeho potomky. Pouze v IE.

Procházení a získávání dat z DOM je přímočarý proces. Mějme následující XML dokument:

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book isbn="9780470109496">Professional Ajax</book>
  <book isbn="0764579088">Professional JavaScript for Web Developers</book>
  <book isbn="0764557599">Professional C#</book>
  <book isbn="1861002025">Professional Visual Basic 6 Databases</book>
</books>
```

Tento jednoduchý XML dokument obsahuje kořenový prvek `<books>` se čtyřmi potomky `<book>`. Použitím tohoto dokumentu jako ukazatele můžete prozkoumat DOM. Strom DOM je založen na vztazích mezi uzly. Jeden uzel může obsahovat jiné uzly, které jsou nazývány jako dceřiné uzly (každý prvek `<book>` je dceřiným uzlem prvku `<books>`). Další uzel může sdílet stejné rodiče jako jiné uzly – v takovém případě se uzly nazývají jako sourozenci (siblings, každý prvek `<book>` je sourozencem jiných prvků `<book>`).

Předpokládejme, že budete chtít získat první prvek `<book>` v dokumentu. Toho snadno dosáhnete pomocí vlastnosti `firstChild`:

```
var oFirstBook = oXmlDom.documentElement.firstChild;
```

Použitím vlastnosti `firstChild` je získána reference na první prvek `<book>` a je přiřazen do proměnné `oFirstBook`, protože je to první dceřiný prvek kořenového prvku `<books>`.

K získání stejného výsledku můžete rovněž použít kolekci `childNodes`:

```
var oFirstBook2 = oXmlDom.documentElement.childNodes[0];
```

Výběr prvního prvku v kolekci `childNodes` (na indexu 0) vrací prvního potomka uzlu (stejně jako použití vlastnosti `firstChild`). Pomocí vlastnosti `length` můžete snadno určit počet potomků, které má daný uzel.

```
var iChildren = oXmlDom.documentElement.childNodes.length;
```

Pokud může mít nějaký uzel potomky, znamená to, že potomci mohou mít rodiče. Vlastnost `ParentNode` vrací rodiče daného uzlu.

```
var oParent = oFirstBook.parentNode;
```

Připomínáme, že `oFirstBook` je první prvek `<book>` v dokumentu. Vlastnost `ParentNode` tohoto uzlu se odkazuje na prvek `<books>`, na `documentElement` dokumentu.

Jednotlivé prvky `<book>` jsou vzájemně sourozenci, protože mají stejného přímého rodiče. Existují dvě vlastnosti pro zpřístupnění těchto sousedních uzlů – `nextSibling` a `previousSibling`. Vlastnost `nextSibling` se odkazuje na následujícího sourozence, zatímco vlastnost `previousSibling` vybírá předcházejícího sourozence:

```
var oSecondBook = oFirstBook.nextSibling;
var oFirstBook2 = oSecondBook.previousSibling;
```

V tomto kódu je druhý prvek `<book>` přiřazen k `oSecondBook`. Proměnná `oFirstBook2` je pak přiřazena k předchozímu sourozenci `oSecondBook`, což má za následek to, že `oFirstBook2` obsahuje stejné hodnoty jako `oFirstBook`. Pokud uzel nemá žádné předcházející nebo následující sourozence, `previousSibling` a `nextSibling` budou `null`.

Když nyní víte, jak procházet skrze hierarchii dokumentu, je dalším krokem provést extrakci z uzlů ve stromu. Například – pro získání textu obsaženého uvnitř třetího prvku `<book>` ("Professional C#") můžete použít vlastnost `text` následujícím způsobem:

```
var sText = oRoot.childNodes[2].text;
```

Vlastnost `text` získá kompletní text, který je obsažený v tomto uzlu. Je to sice patentovaná vlastnost Microsoftu, ale je velice užitečná. Bez ní byste museli text uzlu zpřístupnit takto:

```
var sText = oRoot.childNodes[2].firstChild.nodeValue;
```

Tento kód získává stejné výsledky jako použití vlastnosti `text`. Stejně jako v předchozím příkladě i zde je reference na třetí prvek `<book>` získána pomocí kolekce `childNodes`. Reference na text uzlu z prvku `<book>` je pak získána s použitím `firstChild`, protože textový uzel je pořad DOM uzlu. Samotný text je pak získáván s použitím vlastnosti `nodeValue`, která je pro textový uzel vždy nastavena na jeho obsah.

Výsledky těchto dvou příkladů jsou identické, ačkoliv se vlastnost `text` chová jinak, než je použití vlastnosti `nodeValue` na textový uzel. Vlastnost `text` získává hodnotu všech textových uzlů obsažených v prvku a jeho potomcích, zatímco vlastnost `nodeValue` získává hodnoty pouze ze současného uzlu. Vlastnost `text` je sice užitečná, ale může také vrátit více textu, než by bylo žádoucí.

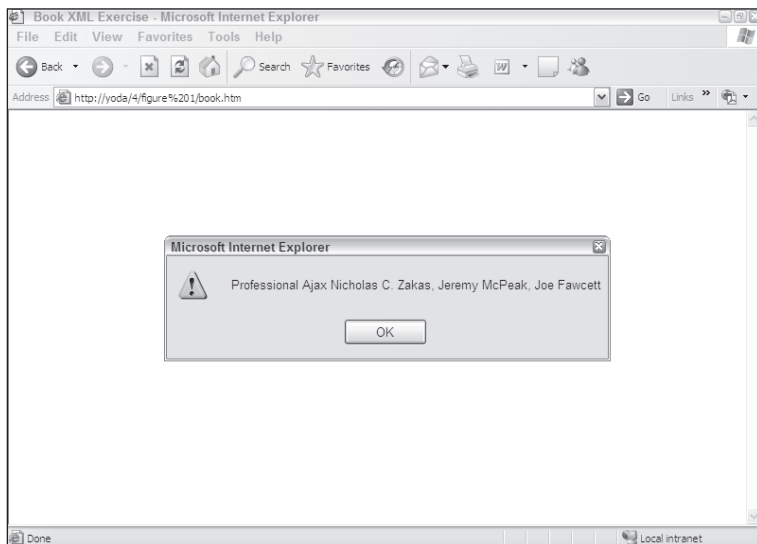
```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book isbn="9780470109496">
    <title>Professional Ajax</title>
    <author>Nicholas C. Zakas, Jeremy McPeak, Joe Fawcett</author>
  </book>
  <book isbn="0764579088">Professional JavaScript for Web Developers</book>
  <book isbn="0764557599">Professional C#</book>
  <book isbn="1861002025">Professional Visual Basic 6 Databases</book>
```

```
</books>
```

Tento nový XML dokument přidává dva nové potomky k prvnímu prvku `<book>` – prvek `<title>`, který obsahuje název knihy, a prvek `<author>`, jenž obsahuje data o autorech. Znovu použijte vlastnost `text`:

```
alert(oFirstBook.text);
```

V tomto kódu není nic nového, ale podívejte se na výsledky, které jsou ukázány na obrázku 6-1.



**Obrázek 6-1.** Výsledek kódu.

Všimněte si, že textové uzly z prvků `<title>` a `<author>` byly získány a zřetězeny. Pokud byste použili `oFirstBook.nodeValue`, vrátilo by se `null`, protože `oFirstBook` není textovým uzlem.

Existuje mnoho metod pro získání uzlů a hodnot z XML uzlů – dvě nejčastěji používané jsou `getAttribute()` a `getElementsByTagName()`.

Metoda `getAttribute()` vezme řetězec argumentů obsahujících název atributu, který chcete získat. Pokud atribut neexistuje, vrátí se hodnota `null`. Pokud použijeme stejný XML dokument, který byl uveden dříve v této kapitole, můžeme pracovat s následujícím kódem:

```
var sAttribute = oFirstBook.getAttribute("isbn");  
alert(sAttribute);
```

Tento kód získává hodnotu atributu `isbn` z prvního prvku `<book>` a přiřazuje ho k proměnné `sAttribute`. Tato hodnota je pak zobrazena pomocí `alert()`.

Metoda `getElementsByTagName()` vrací `NodeList` dceřiných prvků se specifikovaným jménem prvku. Tato metoda vyhledává prvky pouze v daných potomcích uzlu, takže vrácený `NodeList` neobsahuje žádné prvky, které by byly předky nebo potomky předků. Například:



```
var cBooks = oRoot.getElementsByTagName("book");  
alert(cBooks.length);
```

Tento kód získává všechny prvky `<book>` v dokumentu a vrací `NodeList` do `cBooks`. V našem příkladě s XML dokumentem se zobrazí výstražné okno, že byly nalezeny 4 prvky `<book>`. K získání všech potomků prvků zadejte `*` jako parametr do `getElementsByTagName()`. Následovně:

```
var cElements = oRoot.getElementsByTagName("*");
```

V tomto případě kolekce `cElements` obsahuje prvky `<book>` i `<title>` a `<author>`.

## Získávání dat XML v IE

Získávání dat XML je stejně jednoduché jako používání nějaké vlastnosti, v tomto případě vlastnosti `xml`. Tato vlastnost serializuje data XML ze současného uzlu. Serializace je proces konverze objektů do snadno uložitelných a přenositelných formátů. Vlastnost `xml` kompletně zkonvertuje XML do řetězcové reprezentace se jmény prvků, atributů a textem:

```
var sXml = oRoot.xml;  
alert(sXml);
```

Tento kód serializuje data XML počínaje kořenovým prvkem dokumentu. Výsledek je pak předán metodě `alert()`. Část serializovaných dat vypadá nějak takto:

```
<books><book isbn="9780470109496">Professional Ajax</book></books>
```

Serializovaná data je možné načíst do jiného objektu XML DOM, poslat je serverové aplikaci nebo je předat jiné stránce. Serializovaná data XML, která jsou vrácená vlastností `xml`, závisí na aktuálním uzlu. Použití vlastnosti `xml` v uzlu `documentElement` vrací data XML z celého dokumentu, zatímco její použití v prvků `<book>` vrací pouze data obsažená v tomto prvků `<book>`.

*Vlastnost `xml` je pouze pro čtení. Pokud chcete přidat nějaké prvky do dokumentu, budete muset použít metody DOM, což je popsáno dále v této kapitole.*

## Manipulace s DOM v IE

Do této chvíle jste se naučili, jak procházet strukturou DOM, jak z něj vytáhnout informace, a jak převést XML do řetězce. Nyní si ukážeme, jak v DOM přidat, smazat a přemístit uzly.

### Vytváření uzlů

Použitím metod DOM můžete vytvořit několik různých uzlů – nejčastěji používaná je ovšem metoda `createElement()`. Tato metoda přijímá jeden argument: řetězec obsahující název prvku, který má vytvořit. Vráti ukazatel na `XMLDOMElement`:

```
var oNewBook = oXmlDom.createElement("book");  
oXmlDom.documentElement.appendChild(oNewBook);
```

Tento kód vytvoří nový prvek `<book>` a připojí ho k `documentElement` použitím metody `appendChild()`. Tato metoda přidává nové prvky, specifikované jejími argumenty, za poslední dceřiný uzel. Výše uvedený kód přidá do dokumentu prázdný prvek `<book>`, takže je potřeba k němu doplnit nějaký text, jako zde:

```
var oNewBook = oXmlDom.createElement("book");  
var oNewBookText = oXmlDom.createTextNode("Professional .NET 2.0 Generics");  
oNewBook.appendChild(oNewBookText);  
oXmlDom.documentElement.appendChild(oNewBook);
```

Tento kód vytvoří textový uzel pomocí metody `createTextNode()` a připojí ho k nově vytvořenému prvku `<book>` pomocí `appendChild()`. Metoda `createTextNode()` přijímá řetězec jako argument, který specifikuje textový obsah uzlu.

V tomto bodě jsme programově vytvořili nový prvek `<book>`, poskytli mu textový uzel a připojili ho k dokumentu. Aby se tento nový prvek mohl stát sourozeneckým s okolními prvky `<book>`, potřebujeme specifikovat ještě jednu informaci – atribut `isbn`. Vytvoření tohoto atributu je jednoduché – stačí použít metodu `setAttribute()`, která je přístupná pro každý prvek uzlu.

```
var oNewBook = oXmlDom.createElement("book");  
var oNewBookText = oXmlDom.createTextNode("Professional .NET 2.0 Generics");  
oNewBook.appendChild(oNewBookText);  
oNewBook.setAttribute("isbn", "0764559885");  
oXmlDom.documentElement.appendChild(oNewBook);
```

Zvýrazněný řádek kódu v tomto případě vytvoří atribut `isbn` a přiřadí mu hodnotu `0764559885`. Metoda `setAttribute()` pracuje se dvěma řetězci jako argumenty: první argument je název atributu, druhý argument je jeho hodnota. IE poskytuje i jiné metody pro přidávání atributů k prvku. Tyto metody ovšem nenabízejí žádnou významnou výhodu oproti `setAttribute()`, nehledě na to, že vyžadují mnohem více kódování.

### Odstraňování, nahrazování a vkládání uzlů

Pokud umíte přidávat uzly do dokumentu, zdá se být logické, že byste měli být schopni je stejně snadno i odstranit. To uděláte pomocí metody `removeChild()`. Tato metoda přijímá jeden argument – uzel k odstranění. K odstranění prvního prvku `<book>` z dokumentu může být použit následující kód:

```
var oRemovedChild = oRoot.removeChild(oRoot.firstChild);
```

Metoda `removeChild()` vrací dceřiný uzel, který byl odstraněn, takže `oRemovedChild` se odkazuje na odstraněný prvek `<book>`. Pokud máte odkaz (referenci) na starý uzel, můžete jej umístit kamkoliv jinam v dokumentu.