KOMPENDIUM WIEDZY O HTML5!

Apress

HTML5

Przewodnik encyklopedyczny

Adam Freeman

Helion

Tytuł oryginału: The Definitive Guide to HTML5

Tłumaczenie: Maksymilian Gutowski

ISBN: 978-83-246-5081-1

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/htm5pe
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/htm5pe.zip

Printed in Poland.

Spis treści

ROZDZIAŁ 28

Praca z elementami DOM

W poprzednim rozdziale, przy omówieniu funkcji działających na poziomie dokumentu, wspomniałem o niektórych funkcjach obiektu `HTMLElement`. Teraz możemy się już odpowiednio skoncentrować na nim samym. W niniejszym rozdziale przedstawię Ci poszczególne właściwości i metody `HTMLElement` oraz pokażę, jak ich używać. W tabeli 28.1 znajduje się streszczenie rozdziału. Miej w pamięci, że nie wszystkie popularne przeglądarki będą obsługiwać podane tutaj przykłady.

Tabela 28.1. *Streszczenie rozdziału*

Problem	Rozwiązanie	Listing
Pobranie informacji o elemencie.	Użyj właściwości metadanych `HTMLElement`.	28.1
Pobranie lub zdefiniowanie pojedynczego ciągu znaków, zawierającego wszystkie klasy, do których element jest przypisany.	Użyj właściwości `className`.	28.2
Sprawdzenie lub zmodyfikowanie poszczególnych klas elementów.	Użyj właściwości `classList`.	28.3
Pobranie lub zdefiniowanie atrybutów elementu.	Użyj metod `attribute`, `getAttribute`, `setAttribute`, `removeAttribute` i `hasAttribute`.	28.4, 28.6
Pobranie lub zdefiniowanie niestandardowych atrybutów elementu.	Użyj właściwości `dataset`.	28.5
Praca z zawartością tekstową elementu.	Użyj obiektów `Text`.	28.7 – 28.9
Tworzenie i usuwanie elementów.	Użyj metod `document.create*` i `HTMLElement` służących do zarządzania elementami-dziećmi.	28.10
Powielenie elementu.	Użyj metody `cloneNode`.	28.11
Przeniesienie elementu.	Użyj metody `appendChild`.	28.12
Sprawdzenie identyczności dwóch obiektów.	Użyj metody `isSameNode`.	28.13
Sprawdzenie równości dwóch obiektów.	Użyj metody `isEqualNode`.	28.14
Bezpośrednia praca nad fragmentami HTML.	Użyj właściwości `innerHTML` i `outerHTML` oraz metody `insertAdjacentHTML`.	28.15 – 28.17
Wstawienie elementu do bloku tekstu.	Użyj metod `splitText` i `appendChild`.	28.18

Praca z obiektami elementów

Obiekty `HTMLElement` obsługują szereg właściwości, których można używać do odczytywania i modyfikowania danych reprezentowanego elementu. W tabeli 28.2 znajduje się opis tych właściwości.

Tabela 28.2. *Właściwości danych elementów*

Właściwość	Opis	Zwracana wartość
`checked`	Sprawdza lub definiuje obecność atrybutu `checked`.	*`wartość_boolowska`*
`classList`	Pobiera lub definiuje listę klas, do których element przynależy.	`DOMTokenList`
`className`	Pobiera lub definiuje listę klas, do których element przynależy.	*`tekst`*
`dir`	Pobiera lub definiuje wartość atrybutu `dir`.	*`tekst`*
`disabled`	Sprawdza lub definiuje obecność atrybutu `disabled`.	*`wartość_boolowska`*
`hidden`	Sprawdza lub definiuje obecność atrybutu `hidden`.	*`wartość_boolowska`*
`id`	Pobiera lub definiuje wartość atrybutu `id`.	*`tekst`*
`lang`	Pobiera lub definiuje wartość atrybutu `lang`.	*`tekst`*
`spellcheck`	Sprawdza lub definiuje obecność atrybutu `spellcheck`.	*`wartość_boolowska`*
`tabIndex`	Pobiera lub definiuje wartość atrybutu `tabindex`.	*`liczba`*
`tagName`	Zwraca nazwę znacznika (wskazując tym samym rodzaj elementu).	*`tekst`*
`title`	Pobiera lub definiuje wartość atrybutu `title`.	*`tekst`*

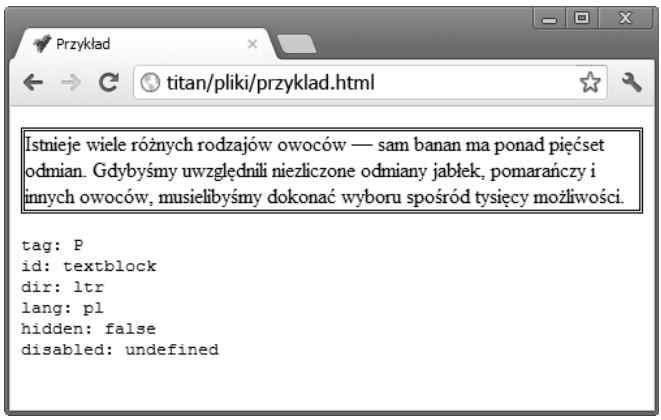
W listingu 28.1 widać przykład zastosowania niektórych z podstawowych właściwości, które podano w liście.

Listing 28.1. *Zastosowanie podstawowych właściwości danych elementów*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {border: medium double black;}
      </style>
   </head>
   <body>

      <p id="textblock" dir="ltr" lang="pl">
         Istnieje wiele różnych rodzajów owoców — sam <span id="banana">banan</span> ma ponad
         pięćset odmian. Gdybyśmy uwzględnili niezliczone odmiany <span id="apple">jabłek</span>,
         <span id="orange">pomarańczy</span> i innych owoców, musielibyśmy dokonać wyboru
         spośród tysięcy możliwości.
      </p>
      <pre id="results"></pre>
      <script>
         var results = document.getElementById("results");
         var elem = document.getElementById("textblock");

         results.innerHTML += "tag: " + elem.tagName + "\n";
```

```
        results.innerHTML += "id: " + elem.id + "\n";
        results.innerHTML += "dir: " + elem.dir + "\n";
        results.innerHTML += "lang: " + elem.lang + "\n";
        results.innerHTML += "hidden: " + elem.hidden + "\n";
        results.innerHTML += "disabled: " + elem.disabled + "\n";
    </script>
</body>
</html>
```

Rezultat zastosowania tych właściwości widać na rysunku 28.1.



Rysunek 28.1. *Pobranie informacji o elemencie*

Praca z klasami

Praca z klasami, do których przypisany jest element, może się odbywać na dwa sposoby. Pierwszy z nich opiera się na wykorzystaniu właściwości `className`, która zwraca listę klas. Klasy można dodawać i usuwać poprzez zmianę wartości ciągu znaków. Przykład odczytania i zmiany klas tym sposobem widać w listingu 28.2.

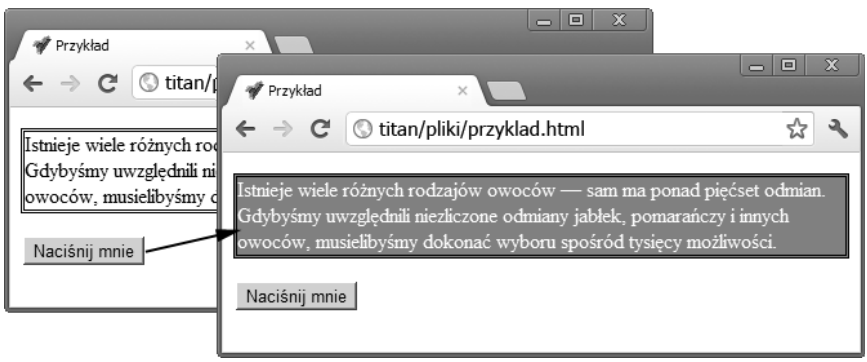
Listing 28.2. *Zastosowanie właściwości className*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {
            border: medium double black;
         }
         p.newclass {
            background-color: grey;
            color: white;
         }
      </style>
   </head>
   <body>

      <p id="textblock" class="fruit numbers">
```

```
        Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy
        uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy
        dokonać wyboru spośród tysięcy możliwości.
    </p>
    <button id="pressme">Naciśnij mnie</button>
    <script>
        document.getElementById("pressme").onclick = function(e) {
            document.getElementById("textblock").className += " newclass";
        };
    </script>
</body>
</html>
```

■ **Wskazówka** Klasy są standardowo używane do przypisywania elementom stylów. O pracy ze stylami w DOM przeczytasz w rozdziale 29.

W powyższym przykładzie naciśnięcie przycisku przywołuje skrypt, który dodaje nową klasę do listy elementu. Zauważ, że musiałem umieścić spację przed każdą wartością dodaną do właściwości `className`, ponieważ przeglądarka spodziewa się listy oddzielonych od siebie spacjami klas. Po wprowadzeniu takiej zmiany przeglądarka nadaje style zgodnie z selektorami klasowymi, wobec czego dokument widocznie zmienia wygląd, tak jak widać to na rysunku 28.2.



Rysunek 28.2. *Zastosowanie właściwości className*

Właściwość `className` jest łatwa w użyciu, kiedy chcesz szybko przypisać element do nowych klas. O wiele trudniej jednak używać jej do jakichkolwiek innych celów, np. do usunięcia klasy. Na szczęście, możesz też użyć właściwości `classList`, która zwraca obiekt `DOMTokenList`. Obiekt ten obsługuje kilka użytecznych metod i właściwości, które pozwalają na zarządzanie listą klas; opisuję je w tabeli 28.3.

Tabela 28.3. *Właściwości i metody DOMTokenList*

Składowa	Opis	Zwracana wartość
`add(<class>)`	Dodaje wskazaną klasę do elementu.	*`void`*
`contains(<class>)`	Zwraca wartość `true`, jeśli element należy do podanej klasy.	*`wartość_boolowska`*
`length`	Zwraca liczbę klas, do których element przynależy.	*`liczba`*
`remove(<class>)`	Usuwa podaną klasę z elementu.	*`void`*
`toggle(<class>)`	Dodaje klasę, jeśli nie jest obecna, a usuwa, jeśli jest obecna.	*`wartość_boolowska`*

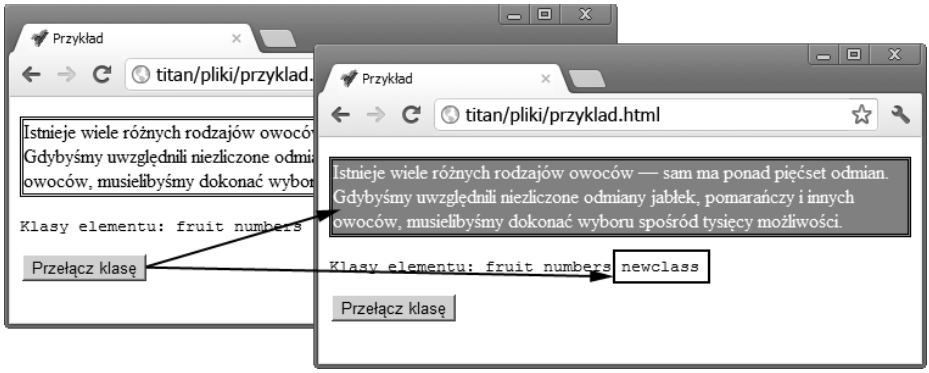
Abstrahując od tych właściwości i metod, masz również możliwość pobierania klas według numeru indeksowego, z wykorzystaniem notacji tablicowej. Przykład wykorzystania obiektu `DOMTokenList` widać w listingu 28.3.

Listing 28.3. *Zastosowanie właściwości classList*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {
            border: medium double black;
         }
         p.newclass {
            background-color: grey;
            color: white;
         }
      </style>
   </head>
   <body>
      <p id="textblock" class="fruit numbers">
         Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy
         uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy
         dokonać wyboru spośród tysięcy możliwości.
      </p>
      <pre id="results"></pre>
      <button id="toggle">Przełącz klasę</button>
      <script>
         var results = document.getElementById("results");
         document.getElementById("toggle").onclick = toggleClass;

         listClasses();
         function listClasses() {
            var classlist = document.getElementById("textblock").classList;
            results.innerHTML = "Klasy elementu: "
            for (var i = 0; i < classlist.length; i++) {
               results.innerHTML += classlist[i] + " ";
            }
         }

         function toggleClass() {
            document.getElementById("textblock").classList.toggle("newclass");
            listClasses();
         }
      </script>
   </body>
   </html>
```

W tym przykładzie funkcja `listClasses` wykorzystuje właściwość `classList` do pobrania i wyliczenia klas, do których przynależy element `p`, wykorzystując przy zwracaniu nazw klas indeksator tablicowy.

Przywoływana po naciśnięciu przycisku funkcja `toggleClass` używa metody `toggle` do dodawania i usuwania klasy `newclass`. Jako że z tą klasą powiązany jest styl, efekt jej zmiany jest widoczny na rysunku 28.3.



Rysunek 28.3. *Wyliczenie i przełączenie klasy*

Praca z atrybutami elementu

Istnieją właściwości obsługujące niektóre z najważniejszych atrybutów globalnych, ale możliwy jest także odczyt i definiowanie poszczególnych atrybutów elementu. W tabeli 28.4 znajduje się opis służących do tego metod i właściwości obsługiwanych przez obiekt `HTMLElement`.

Tabela 28.4. *Właściwości i metody odnoszące się do atrybutów*

Składowa	Opis	Zwracana wartość
`attributes`	Zwraca nadane elementowi atrybuty.	`Attr[]`
`dataset`	Zwraca atrybuty `data-*`.	*`tekst`*`[<nazwa>]`
`getAttribute(<nazwa>)`	Zwraca wartość podanego atrybutu.	*`tekst`*
`hasAttribute(<nazwa>)`	Zwraca wartość `true`, jeśli elementowi przypisany jest podany atrybut.	*`wartość_boolowska`*
`removeAttribute(<nazwa>)`	Usuwa podany atrybut elementu.	*`void`*
`setAttribute(<nazwa>, <wartość>)`	Nadaje atrybut o podanej nazwie i wartości.	*`void`*

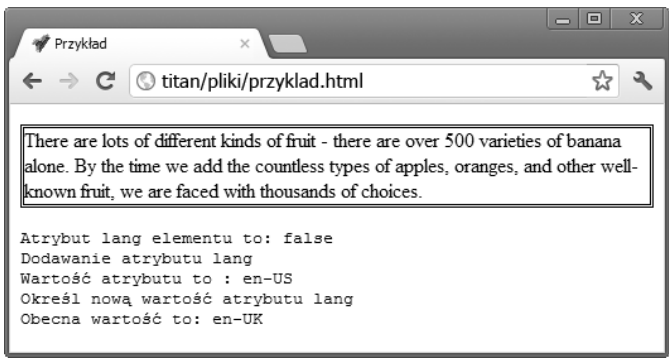
Istnieją cztery proste w użyciu i przewidywalne w działaniu metody, które służą do pracy z atrybutami. W listingu 28.4 widnieje przykład ich zastosowania.

Listing 28.4. *Zastosowanie metod atrybutów*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {border: medium double black;}
      </style>
   </head>
   <body>
      <p id="textblock" class="fruit numbers">
         There are lots of different kinds of fruit - there are over 500 varieties
         of banana alone. By the time we add the countless types of apples, oranges,
         and other well-known fruit, we are faced with thousands of choices.
```

```
      </p>
      <pre id="results"></pre>
      <script>
         var results = document.getElementById("results");
         var elem = document.getElementById("textblock");

         results.innerHTML = "Atrybut lang elementu to: "
            + elem.hasAttribute("lang") + "\n";
         results.innerHTML += "Dodawanie atrybutu lang\n";
         elem.setAttribute("lang", "en-US");
         results.innerHTML += "Wartość atrybutu to : " + elem.getAttribute("lang")
         +"\n";
         results.innerHTML += "Określ nową wartość atrybutu lang\n";
         elem.setAttribute("lang", "en-UK");
         results.innerHTML += "Obecna wartość to: " + elem.getAttribute("lang") + "\n";
      </script>
   </body>
   </html>
```

W tym przykładzie sprawdziłem, dodałem i zmieniłem wartości atrybutu `lang`. Rezultat zastosowania tego skryptu widać na rysunku 28.4.



Rysunek 28.4. *Zastosowanie metod atrybutów*

Praca z atrybutami data-*

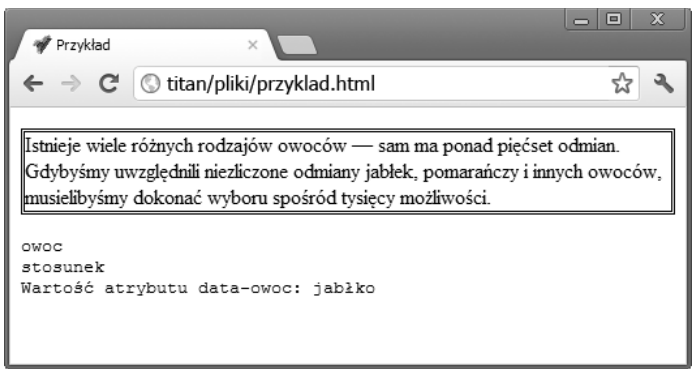
W rozdziale 3. wspomniałem, że HTML5 obsługuje niestandardowe atrybuty, wykorzystując do tego prefiks `data-`. Oznacza to, że możesz stworzyć atrybut w rodzaju `data-mójnowyatrybut`. Z niestandardowymi atrybutami w DOM można pracować z wykorzystaniem właściwości `dataset`, która zwraca tablicę wartości indeksowanych według nazwy owej niestandardowej właściwości. W listingu 28.5 widnieje przykład.

Listing 28.5. *Zastosowanie właściwości dataset*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {border: medium double black;}
      </style>
   </head>
   <body>
```

```
    <p id="textblock" class="fruit numbers" data-owoc="jabłko"
       data-stosunek="pozytywny">
       Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy
       uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy
       dokonać wyboru spośród tysięcy możliwości.
    </p>
    <pre id="results"></pre>
    <script>
       var results = document.getElementById("results");
       var elem = document.getElementById("textblock");

       for (var attr in elem.dataset) {
          results.innerHTML += attr + "\n";
       }

       results.innerHTML += "Wartość atrybutu data-owoc: " + elem.dataset["owoc"];
    </script>
 </body>
 </html>
```

Tablica wartości zwróconych przez właściwość `dataset` nie jest indeksowana według pozycji, tak jak ma to miejsce w przypadku standardowych tablic. Do wyliczania atrybutów `data-*` należy użyć deklaracji `for...in`, tak jak w listingu. Możesz też przywołać wartość według jej nazwy. Zauważ, że należy tu podać jedynie tę część nazwy atrybutu, która znajduje się po członie `data-*`. Jeśli więc chcesz sprawdzić wartość atrybutu `data-owoc`, to powinieneś przywołać wartość `dataset["owoc"]`. Rezultat zastosowania tego skryptu widać na rysunku 28.5.



Rysunek 28.5. *Zastosowanie właściwości dataset*

Praca ze wszystkimi atrybutami

Zbiór wszystkich atrybutów elementu możesz pozyskać przy użyciu właściwości `attributes`, która zwraca tablicę obiektów `Attr`. Właściwości obiektu `Attr` opisane są w tabeli 28.5.

Tabela 28.5. *Właściwości obiektu Attr*

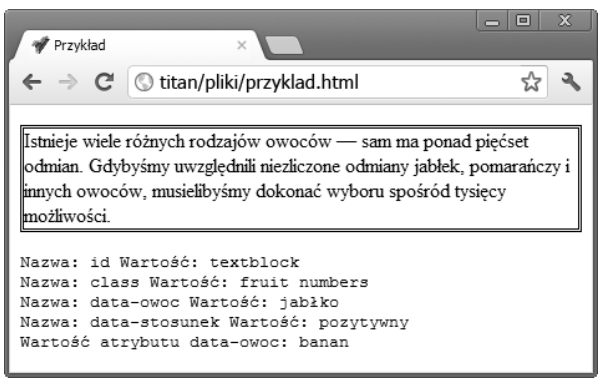
Właściwości	Opis	Zwracana wartość
`name`	Zwraca nazwę atrybutu.	*tekst*
`value`	Zwraca lub definiuje wartość atrybutu.	*tekst*

W listingu 28.6 widnieje przykład zastosowania właściwości `attributes` i obiektu `Attr` do odczytania i zmodyfikowania atrybutów elementu.

Listing 28.6. *Praca z właściwością attributes*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {border: medium double black;}
      </style>
   </head>
   <body>
      <p id="textblock" class="fruit numbers" data-owoc="jabłko"
         data-stosunek="pozytywny">
         Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy
         uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy
         dokonać wyboru spośród tysięcy możliwości.
      </p>
      <pre id="results"></pre>
      <script>
         var results = document.getElementById("results");
         var elem = document.getElementById("textblock");

         var attrs = elem.attributes;

         for (var i = 0; i < attrs.length; i++) {
            results.innerHTML += "Nazwa: " + attrs[i].name + " Wartość: "
               + attrs[i].value + "\n";
         }

         attrs["data-owoc"].value = "banan";

         results.innerHTML += "Wartość atrybutu data-owoc: "
            + attrs["data-owoc"].value;
      </script>
   </body>
   </html>
```

Jak widać w listingu, atrybuty w tablicy obiektów `Attr` indeksowane są według pozycji i nazwy. W tym przykładzie wyliczyłem nazwy i wartości przypisanych elementowi atrybutów, a następnie zmieniłem wartość jednego z nich. Rezultat zastosowania tego skryptu widać na rysunku 28.6.



Rysunek 28.6. *Zastosowanie właściwości attributes*

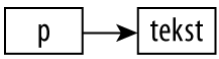
Praca z tekstem

Zawartość tekstową elementu reprezentuje obiekt `Text`, który w modelu dokumentu występuje jako dziecko owego elementu. W listingu 28.7 widnieje element z treścią tekstową.

Listing 28.7. *Element z treścią tekstową*

```
...
<p id="textblock" class="fruit numbers" data-fruit="apple" data-sentiment="like">
Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy uwzględnili
niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy dokonać wyboru spośród
tysięcy możliwości.
</p>
...
```

Reprezentowany w modelu dokumentu element `p` przybiera postać obiektu `HTMLElement`, a jego treść reprezentowana jest przez obiekt `Text`, tak jak widać na rysunku 28.7.



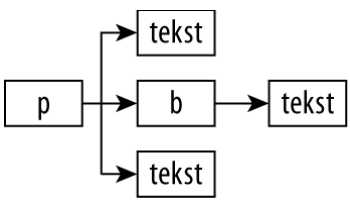
Rysunek 28.7. *Relacja pomiędzy obiektami reprezentującymi element i jego treść*

Wszystkie zawierające tekst dzieci elementu obsługiwane są w ten sam sposób. W listingu 28.8 zamieściłem w akapicie dodatkowy element.

Listing 28.8. *Dodanie elementu do akapitu*

```
...
<p id="textblock" class="fruit numbers" data-fruit="apple" data-sentiment="like">
Istnieje wiele różnych rodzajów owoców — sam ma ponad <b>pięćset</b> odmian. Gdybyśmy
uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy dokonać
wyboru spośród tysięcy możliwości.
</p>
...
```

Wprowadzenie elementu `b` zmienia hierarchię węzłów tak, by przedstawiony w niej był element `p` i jego zawartość, tak jak widać to na rysunku 28.8.



Rysunek 28.8. *Rezultat dodania elementu do akapitu*

Pierwsze dziecko elementu `p` jest obiektem `Text`, który reprezentuje tekst od początku bloku do elementu `b`. Element `b` ma własne dziecko, jakim jest obiekt `Text`, który reprezentuje treść zawartą pomiędzy jego otwierającym a zamykającym znacznikiem. Wreszcie, ostatnie dziecko elementu `p` jest obiektem `Text`, reprezentującym tekst znajdujący się po elemencie `b` — do końca bloku. W tabeli 28.6 znajduje się opis składowych obsługiwanych przez obiekt `Text`.

Niestety, nie ma żadnego wygodnego sposobu na znajdowanie elementów `Text`, poza wyszukiwaniem obiektów elementów-rodziców i przechodzeniem przez ich dzieci. Praca z elementami `Text` jest wobec tego trudniejsza, niż powinna. W listingu 28.9 widnieją niektóre z metod i właściwości elementu `Text`.

Po naciśnięciu elementu `button` wyświetlana jest liczba znaków w pierwszym dziecku `Text` elementu `p`, a jego zawartość zostaje zmieniona metodą `replaceWholeText`.

Tabela 28.6. *Składowe obiektu Text*

Składowa	Opis	Zwraca
`appendData(<tekst>)`	Dodaje określony ciąg znaków na końcu bloku tekstowego.	*void*
`data`	Zwraca lub definiuje tekst.	*tekst*
`deleteData(<pozycja>, <liczba znaków>)`	Usuwa tekst z ciągu znaków. Pierwsza liczba określa pozycję, a druga liczbę znaków do usunięcia.	*void*
`insertData(<pozycja>, <tekst>)`	Zamieszcza podany ciąg znaków na określonej pozycji.	*void*
`length`	Zwraca liczbę znaków.	*liczbę*
`replaceData(<pozycja>, <liczba znaków>, <tekst>)`	Zamienia wskazany zakres tekstu na określony ciąg znaków.	*void*
`replaceWholeText(<tekst>)`	Zamienia cały tekst.	`Text`
`splitText(<liczba>)`	Dzieli element `Text` na dwa osobne elementy we wskazanym miejscu. W punkcie „Wstawienie elementu do bloku tekstu" znajdziesz przykład zastosowania tej metody.	`Text`
`substringData(<pozycja>, <liczba znaków>)`	Zwraca ciąg znaków z tekstu.	*tekst*
`wholeText`	Zwraca tekst.	*tekst*

Listing 28.9. *Praca z obiektami Text*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         p {border: medium double black;}
      </style>
   </head>
   <body>
      <p id="textblock" class="fruit numbers" data-fruit="apple" data-sentiment="like">
         Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy
         uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy
         dokonać wyboru spośród tysięcy możliwości.
      </p>
      <button id="pressme">Naciśnij mnie</button>
      <pre id="results"></pre>
      <script>
         var results = document.getElementById("results");
         var elem = document.getElementById("textblock");

         document.getElementById("pressme").onclick = function() {
            var textElem = elem.firstChild;
            results.innerHTML = "Element zawiera " + textElem.length + " znaków\n";
            textElem.replaceWholeText("Oto nowy ciąg znaków");
         };
      </script>
   </body>
   </html>
```

■ **Ostrzeżenie** Trzeba koniecznie zwrócić uwagę, że przy pracy z tekstem białe znaki nie są kondensowane. Oznacza to, że wszelkie spacje i inne białe znaki służące do ustrukturyzowania kodu HTML są traktowane jako część tekstu.

Modyfikacja modelu

W poprzednich punktach pokazałem Ci, jak używać DOM do modyfikowania pojedynczych elementów. Przy użyciu DOM możesz — na przykład — zmieniać atrybuty i treść tekstową. Jest to możliwe, ponieważ DOM jest aktywnie połączony z dokumentem. Wprowadzenie zmiany w DOM sprawia, że przeglądarka wprowadza odpowiednią zmianę w dokumencie. Dzięki temu połączeniu można nawet zmienić strukturę dokumentu. Można dodawać, usuwać, powielać i kopiować elementy w dowolny sposób. W tym celu wystarczy zmienić hierarchię DOM, a jako że wspomniane połączenie działa na bieżąco, zmiany hierarchii natychmiast stają się widoczne w przeglądarce. W tabeli 28.7 znajduje się opis właściwości i metod służących do zmiany hierarchii DOM.

Tabela 28.7. *Składowe służące do manipulacji strukturą DOM*

Składnik	Opis	Zwracana wartość
`appendChild(`*`HTMLElement`*`)`	Dodaje podanemu elementowi dziecko określonego typu.	`HTMLElement`
`cloneNode(`*`boolean`*`)`	Kopiuje element.	`HTMLElement`
`compareDocumentPosition(`*`HTMLElement`*`)`	Określa względne położenie elementu.	*`liczba`*
`innerHTML`	Zwraca lub definiuje treść elementu.	*`tekst`*
`insertAdjacentHTML(`*`<położenie>`*`,` *`<treść>`*`)`	Zamieszcza kod HTML w miejscu określonym względem elementu.	*`void`*
`insertBefore(`*`<nowyelement>`*`,` *`<element>`*`)`	Zamieszcza podany element przed określonym elementem-dzieckiem.	`HTMLElement`
`isEqualNode(`*`<HTMLElement>`*`)`	Określa, czy podany element jest równy bieżącemu elementowi.	*`wartość_boolowska`*
`isSameNode(`*`HTMLElement`*`)`	Określa, czy podany element jest tożsamy z bieżącym elementem.	*`wartość_boolowska`*
`outerHTML`	Zwraca lub definiuje kod HTML i zawartość elementu.	*`tekst`*
`removeChild(`*`HTMLElement`*`)`	Usuwa określone dziecko bieżącego elementu.	`HTMLElement`
`replaceChild(`*`HTMLElement`*`,` *`HTMLElement`*`)`	Zastępuje dziecko bieżącego elementu.	`HTMLElement`

Przedstawione tutaj właściwości i metody obsługiwane są przez wszystkie obiekty elementów. Ponadto, obiekt `document` obsługuje dwie metody służące do tworzenia nowych elementów. Funkcja ta jest nieodzowna, kiedy chce się dodać treść do dokumentu. Opis tych metod znajduje się w tabeli 28.8.

Tabela 28.8. *Składowe służące do manipulacji strukturą DOM*

Składowa	Opis	Zwraca
`createElement(`*`<znacznik>`*`)`	Tworzy nowy obiekt `HTMLElement` określonego typu.	`HTMLElement`
`createTextNode(`*`<treść>`*`)`	Tworzy nowy obiekt `Text` o podanej treści.	`Text`

Tworzenie i usuwanie elementów

Nowe elementy tworzy się przy użyciu obiektu `document`, a następnie wprowadza je poprzez odnalezienie istniejącego `HTMLElement` i wykorzystanie jednej z opisanych powyżej metod. W listingu 28.10 znajduje się przykład zastosowania tej funkcji.

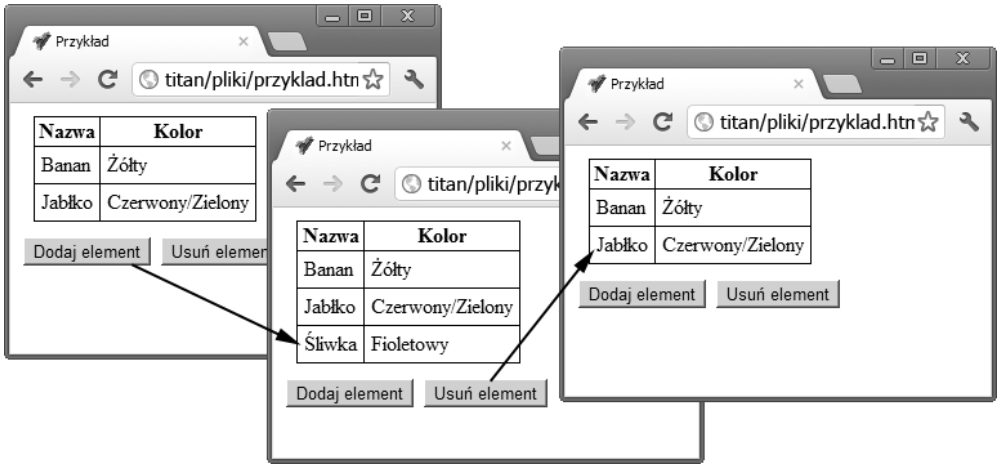
Listing 28.10. *Tworzenie i usuwanie elementów*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
            border-collapse: collapse;
            margin: 10px;
         }
         td { padding: 4px 5px; }
      </style>
   </head>
   <body>
      <table border="1">
         <thead><th>Nazwa</th><th>Kolor</th></thead>
         <tbody id="fruitsBody">
            <tr><td>Banan</td><td>Żółty</td></tr>
            <tr><td>Jabłko</td><td>Czerwony/Zielony</td></tr>
         </tbody>
      </table>

      <button id="add">Dodaj element</button>
      <button id="remove">Usuń element</button>

      <script>
         var tableBody = document.getElementById("fruitsBody");

         document.getElementById("add").onclick = function() {
            var row = tableBody.appendChild(document.createElement("tr"));
            row.setAttribute("id", "newrow");
            row.appendChild(document.createElement("td"))
               .appendChild(document.createTextNode("Śliwka"));
            row.appendChild(document.createElement("td"))
               .appendChild(document.createTextNode("Fioletowy"));
         };

         document.getElementById("remove").onclick = function() {
               var row = document.getElementById("newrow");
               row.parentNode.removeChild(row);
         }
      </script>
   </body>
   </html>
```

Podany w tym przykładzie skrypt wykorzystuje DOM przy dodawaniu i usuwaniu rzędów w elemencie `table` (o którym pisałem w rozdziale 11.). Dodawanie rzędu zaczynam od utworzenia elementu `tr`, którego

następnie używam jako rodzica obiektów `td` i `Text`. Zauważ, że wykorzystałem rezultaty metody do utworzenia łańcucha przywołań i odrobinę uprościłem kod.

Jak widzisz, tworzenie elementów jest pracochłonne. Trzeba utworzyć element, powiązać go z rodzicem, a następnie powtórzyć proces przy każdym elemencie-dziecku bądź obiekcie `Text`. Usuwanie elementów również jest mało poręczne. W tym celu trzeba znaleźć element, przejść do elementu-rodzica, a następnie użyć metody `removeChild`. Rezultat zastosowania tego skryptu widać na rysunku 28.9.



Rysunek 28.9. *Wykorzystanie DOM do tworzenia i usuwania elementów*

Powielanie elementów

Metoda `cloneNode` służy do powielania istniejących elementów. Jest ona wygodna o tyle, że pozwala na uniknięcie tworzenia elementów od podstaw. W listingu 28.11 znajduje się przykład zastosowania tej techniki.

Listing 28.11. *Powielanie elementów*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
            border-collapse: collapse;
            margin: 10px;
         }
         td { padding: 4px 5px; }
      </style>
   </head>
   <body>
      <table border="1">
         <thead><tr><th>Pomnóż</th><th>Wynik</th></tr></thead>
         <tbody id="fruitsBody">
            <tr><td class="sum">1 x 1</td><td class="result">1</td></tr>
         </tbody>
```

```
      </table>

      <button id="add">Dodaj rząd</button>
      <script>
         var tableBody = document.getElementById("fruitsBody");

         document.getElementById("add").onclick = function() {
            var count = tableBody.getElementsByTagName("tr").length + 1;

            var newElem = tableBody.getElementsByTagName("tr")[0].cloneNode(true);
            newElem.getElementsByClassName("sum")[0].firstChild.data = count
               + " + " + count;
            newElem.getElementsByClassName("result")[0].firstChild.data =
               count * count;

            tableBody.appendChild(newElem);
         };
      </script>
   </body>
   </html>
```

W tym przykładzie powieliłem znajdujący się w tabeli rząd. Boolowski argument metody `cloneNode` określa, czy elementy-dzieci również mają być powielone. W tym przypadku podałem wartość `true`, ponieważ chciałem, by elementy `td` zawarte w elemencie `tr` utworzyły strukturę nowego rzędu.

■ **Wskazówka** Zauważ, jak niezręcznie wygląda określanie tekstu komórek tabeli w tym przykładzie. Praca z obiektami `Text` jest naprawdę męcząca. Z prostszym podejściem zapoznasz się w punkcie „Praca z fragmentami HTML" w dalszej części tego rozdziału.

Przenoszenie elementów

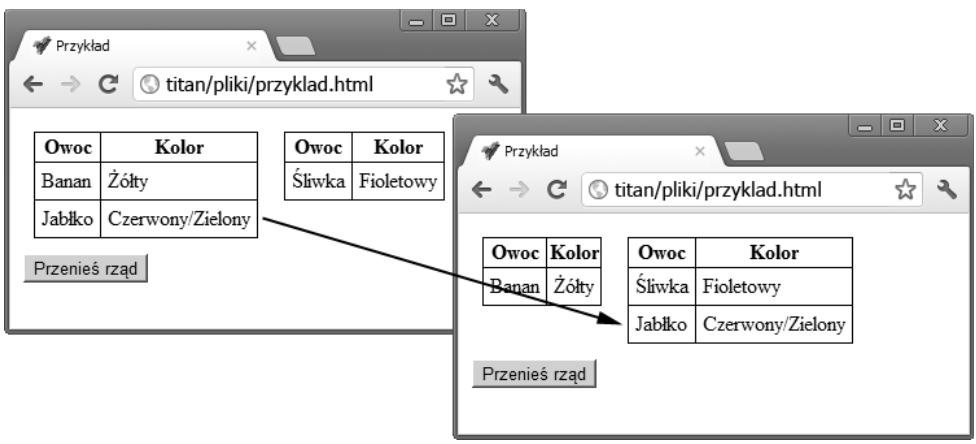
Przenoszenie elementów z jednej części dokumentu do drugiej polega zwyczajnie na powiązaniu elementu z jego nowym rodzicem. Nie musisz przenosić elementu z jego pozycji wyjściowej. W listingu 28.12 widnieje przykład przeniesienia rzędu z jednej tabeli do drugiej.

Listing 28.12. *Przenoszenie elementów*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
            border-collapse: collapse;
            margin: 10px;
            float: left;
         }
         td { padding: 4px 5px; }
         p { clear:left; }
      </style>
   </head>
```

```
<body>
   <table border="1">
      <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
      <tbody>
         <tr><td>Banan</td><td>Żółty</td></tr>
         <tr id="apple"><td>Jabłko</td><td>Czerwony/Zielony</td></tr>
      </tbody>
   </table>

   <table border="1">
      <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
      <tbody id="fruitsBody">
         <tr><td>Śliwka</td><td>Fioletowy</td></tr>
      </tbody>
   </table>

   <p>
      <button id="move">Przenieś rząd</button>
   </p>
   <script>
      document.getElementById("move").onclick = function() {
         var elem = document.getElementById("apple");
         document.getElementById("fruitsBody").appendChild(elem);
      };
   </script>
</body>
</html>
```

Po naciśnięciu przycisku skrypt przenosi element `tr` z atrybutem `id` o wartości `apple` i używa metody `appendChild` na elemencie `tbody` o atrybucie `id` o wartości `fruitsBody`. Rezultatem jest przeniesienie rzędu z jednej tabeli do innej, co widać na rysunku 28.10.



Rysunek 28.10. *Przeniesienie elementu z jednej części dokumentu do innej*

Porównywanie obiektów elementów

Obiekty elementów można porównywać na dwa sposoby. Po pierwsze, wystarczy sprawdzić, czy reprezentują ten sam element, co robi się przy użyciu metody `isSameNode`. Pozwala to na porównanie obiektów, które zostały pozyskane w różnych kwerendach, tak jak widać to w listingu 28.13.

Listing 28.13. *Porównanie obiektów elementów*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
            border-collapse: collapse;
         }
         td { padding: 4px 5px; }
      </style>
   </head>
   <body>
      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody id="fruitsBody">
            <tr id="plumrow"><td>Śliwka</td><td>Fioletowy</td></tr>
         </tbody>
      </table>
      <pre id="results"></pre>
      <script>

         var elemByID = document.getElementById("plumrow");
         var elemByPos
            = document.getElementById("fruitsBody").getElementsByTagName("tr")[0];

         if (elemByID.isSameNode(elemByPos)) {
            document.getElementById("results").innerHTML = "Obiekty są tożsame";
         }

      </script>
   </body>
   </html>
```

Podany w tym przykładzie skrypt znajduje obiekty elementów na dwa różne sposoby: szukając konkretnego `id` oraz przeszukując element-rodzica według rodzaju znacznika. Metoda `isSameNode` zwraca wartość `true` przy porównywaniu tych obiektów, ponieważ reprezentują ten sam element.

Zamiast tego można też sprawdzić, czy obiekty są sobie równe, co robi się metodą `isEqualNode`. Elementy są równe, kiedy są jednakowego typu, mają te same wartości atrybutów, a także wszystkie ich dzieci są równe i występują w tej samej kolejności. W listingu 28.14 znajduje się przykład pary równych elementów.

Listing 28.14. *Praca z równymi elementami*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
            border-collapse: collapse;
```

```
            margin: 2px 0px;
      }
      td { padding: 4px 5px; }
      </style>
   </head>
   <body>
      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody>
            <tr class="plumrow"><td>Śliwka</td><td>Fioletowy</td></tr>
         </tbody>
      </table>

      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody>
            <tr class="plumrow"><td>Śliwka</td><td>Fioletowy</td></tr>
         </tbody>
      </table>

      <pre id="results"></pre>
      <script>
         var elems = document.getElementsByClassName("plumrow");

         if (elems[0].isEqualNode(elems[1])) {
            document.getElementById("results").innerHTML = "Elementy są równe";
         } else {
            document.getElementById("results").innerHTML = "Elementy NIE SĄ równe";
         }
      </script>
   </body>
   </html>
```

W tym przykładzie obydwa elementy `tr` są równe, pomimo że są osobnymi elementami w różnych częściach dokumentu. Gdybym zmienił którykolwiek z atrybutów lub treść elementu-dziecka `td`, elementy nie byłyby już równe.

Praca z fragmentami HTML

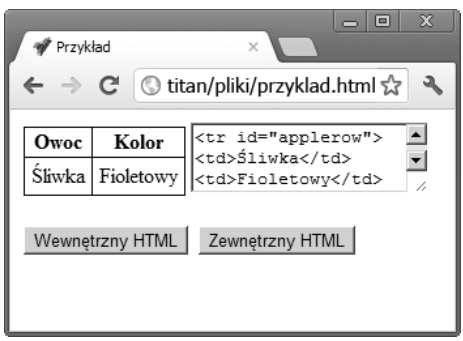
Właściwości `innerHTML` i `outerHTML` wraz z metodą `insertAdjacentHTML` są wygodnymi skrótami składniowymi, które pozwalają na pracę z fragmentami kodu HTML. Dzięki temu można uniknąć tworzenia rozbudowanych hierarchii składających się z obiektów elementów i obiektów tekstowych. W listingu 28.15 widnieje przykład zastosowania właściwości `innerHTML` i `outerHTML` do pobierania kodu HTML z elementów.

Listing 28.15. *Zastosowanie właściwości innerHTML i outerHTML*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
```

```
            border-collapse: collapse;
            margin: 5px 2px;
            float: left;
      }
      td { padding: 4px 5px; }
      p {clear: left};
      </style>
   </head>
   <body>
      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody>
            <tr id="applerow"><td>Śliwka</td><td>Fioletowy</td></tr>
         </tbody>
      </table>
      <textarea rows="3" id="results"></textarea>
      <p>
         <button id="inner">Wewnętrzny HTML</button>
         <button id="outer">Zewnętrzny HTML</button>
      </p>
      <script>
         var results = document.getElementById("results");
         var row = document.getElementById("applerow");

         document.getElementById("inner").onclick = function() {
            results.innerHTML = row.innerHTML;
         };

         document.getElementById("outer").onclick = function() {
            results.innerHTML = row.outerHTML;
         }
      </script>
   </body>
   </html>
```

Właściwość `outerHTML` zwraca ciąg znaków, w którym znajduje się definiujący element kod HTML oraz kod wszystkich jego dzieci. Właściwość `innerHTML` zwraca kod HTML samych dzieci. W tym przykładzie utworzyłem dwa przyciski, które służą do wyświetlania „wewnętrznego" i „zewnętrznego" kodu HTML rzędu tabeli. Treść wyświetlana jest w elemencie `textarea`, tak by przeglądarka traktowała zwracane przez te właściwości ciągi znaków jako tekst, a nie kod HTML dokumentu. Rezultat zastosowania skryptu widać na rysunku 28.11.



Rysunek 28.11. *Wyświetlanie właściwości outerHTML rzędu tabeli*

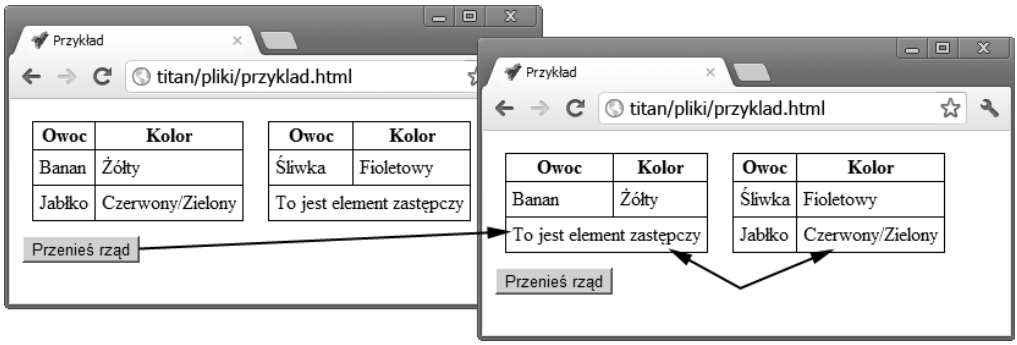
Zmiana struktury dokumentu

Właściwości `outerHTML` i `innerHTML` używa się także do zmiany struktury dokumentu. Właściwości `innerHTML` użyłem w wielu przykładach przedstawionych w tej części książki. Jest to wygodny sposób ustawiania zawartości elementów, ponieważ umożliwia podawanie treści tekstowej bez konieczności tworzenia elementów `Text`. W listingu 28.16 widnieje przykład zastosowania tych właściwości w ramach modyfikowania modelu dokumentu.

Listing 28.16. *Modyfikacja modelu dokumentu*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
      <style>
         table {
            border: solid thin black;
            border-collapse: collapse;
            margin: 10px;
            float: left;
      }
      td { padding: 4px 5px; }
      p { clear: left; }
      </style>
   </head>
   <body>
      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody>
            <tr><td>Banan</td><td>Żółty</td></tr>
            <tr id="apple"><td>Jabłko</td><td>Czerwony/Zielony</td></tr>
         </tbody>
      </table>

      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody id="fruitsBody">
            <tr><td>Śliwka</td><td>Fioletowy</td></tr>
            <tr id="targetrow"><td colspan="2">To jest element zastępczy</td></tr>
         </tbody>
      </table>

      <p>
         <button id="move">Przenieś rząd</button>
      </p>
      <script>
         document.getElementById("move").onclick = function() {
            var source = document.getElementById("apple");
               var target = document.getElementById("targetrow");
            target.innerHTML = source.innerHTML;
            source.outerHTML = '<tr id="targetrow"><td colspan="2">' +
               'To jest element zastępczy</td>';
         };
      </script>
   </body>
   </html>
```

W tym przykładzie użyłem właściwości `innerHTML` do zdefiniowania elementów-dzieci rzędu tabeli oraz `outerHTML` do zastąpienia elementu w wierszu. Właściwości te działają, opierając się na ciągach znaków, co oznacza, że fragmenty HTML pozyskuje się przez odczytanie wartości właściwości lub tworzenie ciągów znakowych od podstaw, tak jak widać to w listingu. Rezultat widać na rysunku 28.12.



Rysunek 28.12. *Zastosowanie właściwości innerHTML i outerHTML*

Wstawianie fragmentów HTML

Właściwości `innerHTML` i `outerHTML` przydają się przy zastępowaniu istniejących elementów, ale jeśli chcesz użyć fragmentu HTML do wprowadzenia nowych elementów, to musisz skorzystać z metody `insertAdjacentHTML`. Metoda ta używa dwóch argumentów — pierwszy to wartość z tabeli 28.9, która wskazuje, gdzie fragment powinien być wstawiony względem bieżącego dokumentu, a drugi to właśnie fragment, który ma być wstawiony.

Tabela 28.9. *Wartości położenia metody insertAdjacentHTML*

Wartość	Opis
`afterbegin`	Wstawia fragment jako pierwsze dziecko bieżącego elementu.
`afterend`	Wstawia fragment bezpośrednio po bieżącym elemencie.
`beforebegin`	Wstawia fragment bezpośrednio przed bieżącym elementem.
`beforeend`	Wstawia fragment jako ostatnie dziecko bieżącego elementu.

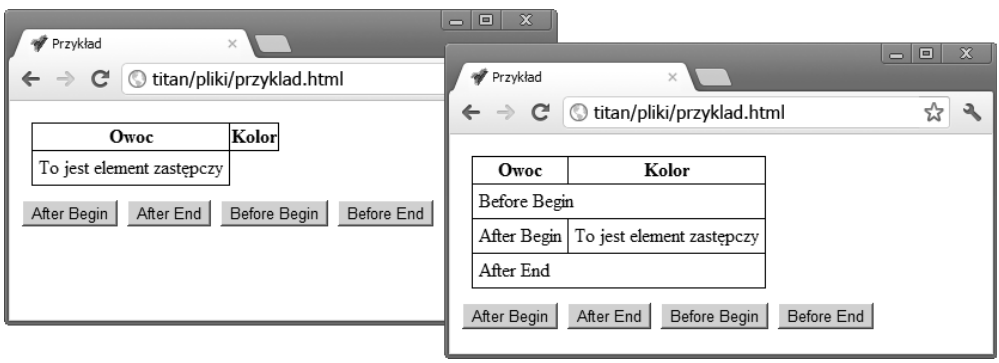
W listingu 28.17 widnieje przykład wykorzystania metody `insertAdjacentHTML` przy wstawianiu fragmentów kodu HTML do rzędu tabeli i dookoła niego.

Listing 28.17. *Zastosowanie metody insertAdjacentHTML*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
   </head>
   <body>
      <table border="1">
         <thead><tr><th>Owoc</th><th>Kolor</th></tr></thead>
         <tbody id="fruitsBody">
            <tr id="targetrow"><td>To jest element zastępczy</td></tr>
         </tbody>
```

```
    </table>

    <p>
       <button id="ab">After Begin</button>
       <button id="ae">After End</button>
       <button id="bb">Before Begin</button>
       <button id="be">Before End</button>
    </p>
    <script>
       var target = document.getElementById("targetrow");
       var buttons = document.getElementsByTagName("button");
       for (var i = 0; i < buttons.length; i++) {
          buttons[i].onclick = handleButtonPress;
       }

       function handleButtonPress(e) {
          if (e.target.id == "ab") {
             target.insertAdjacentHTML("afterbegin", "<td>After Begin</td>");
       } else if (e.target.id == "be") {
          target.insertAdjacentHTML("beforeend", "<td>Before End</td>");
       } else if (e.target.id == "bb") {
          target.insertAdjacentHTML("beforebegin",
             "<tr><td colspan='2'>Before Begin</td></tr>");
       } else {
          target.insertAdjacentHTML("afterend",
             "<tr><td colspan='2'>After End</td></tr>");
       }
    }

    </script>
</body>
</html>
```

W tym przykładzie użyłem różnych wartości położenia, by zaprezentować zamieszczanie fragmentów HTML w różnych miejscach. Najlepiej poeksperymentować z nimi w przeglądarce, aczkolwiek ogólny efekt widać na rysunku 28.13.



Rysunek 28.13. *Wstawianie fragmentów HTML do dokumentu*

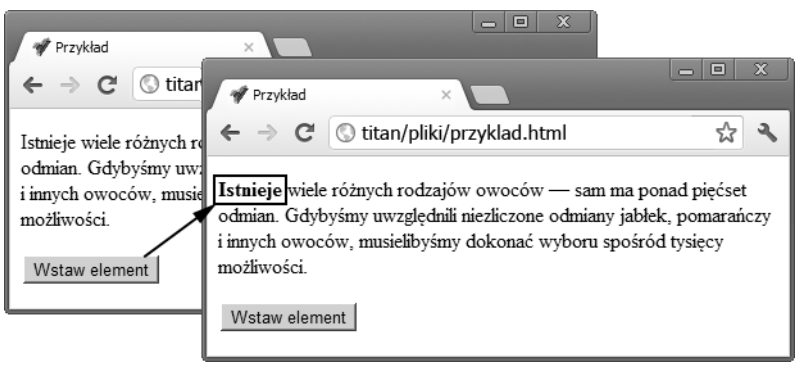
Wstawienie elementu do bloku tekstu

Istotną wariacją modyfikacji modelu jest możliwość dodania elementu do bloku tekstu reprezentowanego obiektem `Text`. W listingu 28.18 widnieje tego przykład.

Listing 28.18. *Wstawienie elementu do bloku tekstu*

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Przykład</title>
      <meta name="author" content="Adam Freeman"/>
      <meta name="description" content="Prosty przykład"/>
      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
   </head>
   <body>
      <p id="textblock">
         Istnieje wiele różnych rodzajów owoców — sam ma ponad pięćset odmian. Gdybyśmy
         uwzględnili niezliczone odmiany jabłek, pomarańczy i innych owoców, musielibyśmy
         dokonać wyboru spośród tysięcy możliwości.
      </p>
      <p>
         <button id="insert">Wstaw element</button>
      </p>
      <script>
         document.getElementById("insert").onclick = function() {
         var textBlock = document.getElementById("textblock");
         textBlock.firstChild.splitText(10);
         var newText = textBlock.childNodes[1].splitText(8).previousSibling;
         textBlock.insertBefore(document.createElement("b"),
            newText).appendChild(newText);
      }
      </script>
   </body>
   </html>
```

W tym przykładzie wykonałem niebanalne zadanie, jakim jest przeniesienie wyrazu z istniejącego bloku tekstowego i zrobienie z niego dziecka nowego elementu `b`. Podobnie jak w poprzednich przykładach, praca z modelem wiąże się z tworzeniem dość rozbudowanego kodu. Rezultat widać na rysunku 28.14.



Rysunek 28.14. *Zamieszczenie elementu w bloku tekstu*

Podsumowanie

W tym rozdziale przedstawiłem funkcje obiektów `HTMLElement` i `Text`, które — kolejno — reprezentują elementy i treść w dokumentach HTML. Dowiedziałeś się, jak pobierać informacje o elementach z obiektów, jak pracować z treścią tekstową oraz jak wykorzystywać możliwości DOM, by dodawać, modyfikować,

powielać, przenosić i usuwać elementy. Praca z DOM może wymagać tworzenia rozbudowanych skryptów, ale aktywne połączenie między modelem obiektowym a wyglądem dokumentu sprawia, że jest to warte wysiłku.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Y

Z

Ź

Ż

POZNAJ I WYKORZYSTAJ POTENCJAŁ HTML5!

HTML5 to hit ostatnich miesięcy. Jego możliwości robią ogromne wrażenie na projektantach stron internetowych, a użytkownicy na tym korzystają. Usługi geolokalizacyjne, zaawansowane funkcje graficzne oraz rozbudowana obsługa multimediów — to tylko niektóre z atutów HTML5. Można śmiało i bez przesady powiedzieć, że język ten będzie gorącym tematem jeszcze przez długi czas. Dlatego warto już dziś sprawdzić, jak wykorzystać jego potencjał.

Dzięki tej książce szybko opanujesz zasady HTML5. Już wkrótce bez najmniejszego problemu będziesz korzystał z nowych znaczników canvas, audio i video. Ponadto nauczysz się stosować mechanizmy do przechowywania danych w lokalnych zasobach przeglądarki oraz sprawdzisz, gdzie w danej chwili znajduje się użytkownik. Autor książki kładzie nacisk na poprawność semantyczną tworzonego kodu. Jest to niezwykle istotne w obecnych czasach, gdy Twoje dzieło będzie oglądane na przeróżnych urządzeniach – zarówno stacjonarnych, jak i mobilnych. Książka ta jest rewelacyjnym źródłem informacji o językach HTML5, CSS oraz JavaScript. Musisz ją mieć!

HTML5 to nowe możliwości:

- obsługa plików multimedialnych,
- zaawansowane funkcje graficzne,
- usługi geolokalizacyjne,
- możliwość pracy bez połączenia z siecią,
- dostępność na różnych urządzeniach!

Apress

Nr katalogowy: 11939

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900 | 0 601 339900

helion.pl księgarnia internetowa

Sprawdź najnowsze promocje: http://helion.pl/promocje
Książki najchętniej czytane: http://helion.pl/bestsellery
Zamów informacje o nowościach: http://helion.pl/nowosci

Helion

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Cena 129,00 zł
ISBN 978-83-246-5081-1
9 788324 650811

Informatyka w najlepszym wydaniu