JavaScript. Nieoficjalny podręcznik

Autor: David Sawyer McFarland
Tłumaczenie: Tomasz Walczak
ISBN: 978-83-246-2166-8
Tytuł oryginału: JavaScript: The Missing Manual
Format: 168×237, stron: 520

Wykorzystaj możliwości JavaScript!

- Jak rozpocząć przygodę z JavaScript?
- Jak dynamicznie modyfikować strony WWW?
- Jak wykorzystać możliwości technologii AJAX?

JavaScript to obiektowy język programowania, który tchnął życie w świat statycznych stron WWW. Sprawdzanie poprawności formularzy, animacje, interaktywność to tylko niektóre z możliwości tego języka. Jednak to, co ostatecznie ugruntowało jego pozycję, to technologia AJAX. Dzięki niej strony internetowe mogą zachowywać się tak, jak standardowe aplikacje, znane z codziennej pracy. Warto zastanowić się, czy wszystkie możliwości JavaScript zostały już odkryte? Może to właśnie Ty zastosujesz go w nowatorski sposób? Pewne jest, że ta książka Ci w tym pomoże!

Książka „JavaScript. Nieoficjalny podręcznik" stanowi idealne źródło informacji na temat programowania w języku JavaScript. Na samym początku poznasz jego składnię, typy danych oraz wszelkie elementy, które pozwolą Ci na swobodną pracę. Po zaznajomieniu się z podstawami przejdziesz do bardziej zaawansowanych tematów. Nauczysz się dynamicznie modyfikować strony WWW, obsługiwać zdarzenia, wykorzystywać bibliotekę jQuery czy też w efektowny sposób prezentować zdjęcia. Ponadto zdobędziesz wiedzę na temat budowania przejrzystych formularzy, tworzenia łatwego w obsłudze interfejsu oraz sposobów wykorzystywania możliwości technologii AJAX. Nie da się ukryć, że dzięki tej książce Twoje strony WWW zyskają na atrakcyjności!

- Narzędzie do programowania w JavaScript
- Podstawy HTML oraz CSS
- Typowe konstrukcje języka JavaScript
- Typy danych
- Wykorzystanie zmiennych
- Logika i struktury sterujące
- Wykorzystanie modelu DOM
- Podstawy pracy z biblioteką jQuery
- Obsługa zdarzeń
- Efekty związane z rysunkami
- Wykorzystanie wtyczki lightBox
- Tworzenie przejrzystych i inteligentnych formularzy
- Kontrola poprawności wprowadzanych danych
- Wykorzystanie technologii AJAX
- Zaawansowane zagadnienia, związane z programowaniem w JavaScript
- Diagnoza i rozwiązywanie typowych problemów

Twórz atrakcyjne witryny WWW, korzystając z JavaScript!

Spis treści

ROZDZIAŁ 5

Dynamiczne modyfikowanie stron WWW

JavaScript umożliwia modyfikowanie stron WWW na oczach użytkownika. Za pomocą kodu w tym języku można błyskawicznie dodawać rysunki i tekst, usuwać fragmenty stron oraz zmieniać wygląd poszczególnych elementów. Dynamiczne modyfikowanie stron to podstawowa cecha najnowszej odmiany stron WWW opartych na języku JavaScript. Jedna z takich witryn, Google Maps (*http://maps.google.com/*), udostępnia mapy całego świata. Kiedy użytkownik zbliży obraz lub go przesunie, witryna zaktualizuje stronę bez konieczności pobierania jej z serwera. Z kolei w serwisie Netflix (*www.netflix.com*) na stronie pojawia się „dymek" ze szczegółowymi informacjami na temat filmu (rysunek 5.1). W obu tych witrynach JavaScript modyfikuje kod HTML pobrany przez przeglądarkę.

W czterech pierwszych rozdziałach książki poznałeś podstawy języka JavaScript — słowa kluczowe, różne mechanizmy i składnię. Teraz, kiedy umiesz już napisać prosty program w języku JavaScript i dodać go do strony, dowiesz się, dlaczego język ten jest tak popularny. W rozdziale tym i następnym (poświęconym zdarzeniom języka JavaScript) zobaczysz, jak tworzyć fantastyczne interaktywne efekty znane z nowoczesnych stron WWW.

Modyfikowanie stron WWW — wstęp

W tym rozdziale dowiesz się, jak zmodyfikować stronę za pomocą kodu JavaScript. Zobaczysz, jak dodać do dokumentu nową treść, znaczniki i atrybuty HTML, a także jak zmienić tekst i tagi zapisane już w dokumencie. Nauczysz się generować nowy kod HTML i modyfikować fragmenty obecne na stronie.

Rysunek 5.1. *Za pomocą języka JavaScript można wyświetlać informacje, kiedy są potrzebne, co upraszcza przeglądanie i czytanie stron. W witrynie Netflix.com opisy pojawiają się dopiero po najechaniu kursorem myszy na miniaturę plakatu do filmu pod jego tytułem*

Dodawanie paska narzędzi z dynamicznym menu, wyświetlanie pokazów slajdów opartych na języku JavaScript lub zmiana koloru co drugiego wiersza tabeli (co zrobiłeś w przykładzie z rozdziału 1.) — wszystkie te operacje wymagają zmodyfikowania treści albo kodu HTML strony. Proces wprowadzania zmian składa się z dwóch etapów. Są to:

1. **Identyfikacja elementu na stronie.**

   Element to dowolny znacznik na stronie. Zanim go zmodyfikujesz, musisz go *zidentyfikować* za pomocą kodu JavaScript (w tym rozdziale nauczysz się to robić). Na przykład aby dodać kolor do wiersza tabeli, trzeba najpierw zidentyfikować ten wiersz. Aby wyświetlić menu po umieszczeniu kursora nad przyciskiem, trzeba znaleźć ten przycisk. Nawet jeśli chcesz użyć kodu JavaScript do dodania tekstu w dolnej części strony, musisz zidentyfikować odpowiedni znacznik, aby umieścić tekst przed, w lub za nim.

2. **Zrobienie czegoś ze znalezionym elementem.**

   To prawda, „zrobienie czegoś" to bardzo ogólne stwierdzenie. Wynika to z tego, że z elementem można *zrobić* niezwykle wiele rzeczy, aby zmodyfikować wygląd lub działanie strony. Większość tej książki opisuje przeprowadzanie różnych operacji na różnych częściach stron WWW. Oto kilka możliwości:

- **Dodawanie i usuwanie atrybutu `class`**. W przykładzie ze strony 42 użyłeś kodu JavaScript do przypisania klasy do co drugiego wiersza tabeli. Kod JavaScript nie „koloruje" komórek, a jedynie określa ich klasę. Następnie przeglądarka używa informacji z arkusza CSS do zmiany wyglądu wierszy.
- **Zmienianie właściwości elementów**. Na przykład aby za pomocą animacji przenieść element `<div>` wzdłuż strony, należy wielokrotnie zmienić jego pozycję.
- **Dodawanie nowej treści**. Jeśli użytkownik nieprawidłowo wypełni pole formularza, można wyświetlić komunikat o błędzie, na przykład „Podaj adres e-mail". Wymaga to dodania odpowiedniego tekstu w pobliżu pola formularza.
- **Usuwanie elementów**. W witrynie Netflix (rysunek 5.1) „dymek" znika po przeniesieniu kursora myszy poza plakat do filmu. Technika ta polega na usunięciu elementu ze strony za pomocą kodu JavaScript.
- **Pobieranie informacji z elementu**. Czasem potrzebne są informacje na temat zidentyfikowanego znacznika. Na przykład aby przeprowadzić walidację pola tekstowego, trzeba je znaleźć, a następnie sprawdzić, jaki tekst wpisał użytkownik, czyli ustalić wartość odpowiedniego pola.

Ten rozdział dotyczy głównie pierwszego etapu — identyfikacji elementu na stronie. Aby zrozumieć, jak wyszukiwać i modyfikować fragmenty stron za pomocą kodu JavaScript, trzeba najpierw poznać *obiektowy model dokumentu* (ang. *Document Object Model* — DOM).

Wprowadzenie do modelu DOM

Kiedy przeglądarka wczyta plik HTML, wyświetla jego zawartość na ekranie (oczywiście po zastosowaniu stylów CSS). Jednak oprócz tego przeglądarki używają znaczników, atrybutów i treści pliku do tworzenia oraz zapisywania „modelu" kodu HTML. Oznacza to, że przeglądarka zapamiętuje znaczniki HTML, ich atrybuty i kolejność pojawiania się w pliku. Taka reprezentacja strony to *obiektowy model dokumentu*, w skrócie DOM.

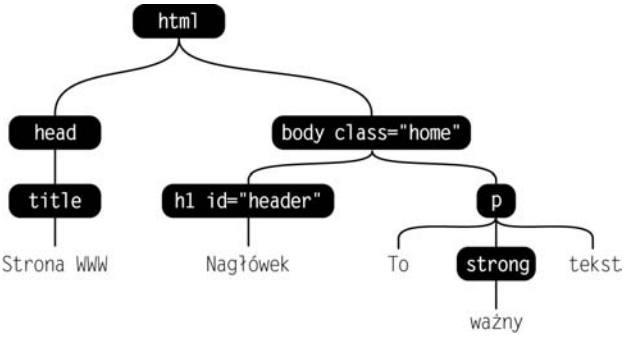
Model DOM udostępnia informacje potrzebne w kodzie JavaScript do uzyskania dostępu do elementów strony. DOM zapewnia też narzędzia potrzebne do nawigowania po kodzie HTML dokumentu, modyfikowania go i dodawania do niego nowych elementów. Sam model DOM nie jest częścią języka JavaScript. Jest to standard opracowany przez organizację W3C (ang. *World Wide Web Consortium*) i przyjęty przez większość producentów przeglądarek. Model DOM umożliwia komunikację z kodem HTML strony i modyfikowanie go z poziomu kodu JavaScript.

Aby zobaczyć, jak działa model DOM, przyjrzyj się bardzo prostej stronie WWW:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
<head>
<title>Strona WWW</title>
</head>
<body class="home">
```

```
<h1 id="header">Nagłówek</h1>
<p>To <strong>ważny</strong> tekst</p>
</body>
</html>
```

Na tej stronie (podobnie jak na wszystkich innych) niektóre znaczniki obejmują inne tagi. Na przykład znacznik `<html>` zawiera wszystkie pozostałe tagi, a w znaczniku `<body>` znajdują się tagi i tekst widoczne w oknie przeglądarki. Relacje między znacznikami można przedstawić w formie podobnej do drzewa genealogicznego (rysunek 5.2). Tag `<html>` to „korzeń" tego drzewa. Można powiedzieć, że jest prapraprapradziadem wszystkich pozostałych znaczników strony. Inne tagi to różne „gałęzie" drzewa. Są to na przykład znaczniki `<head>` i `<body>`, które obejmują następne tagi.



Rysunek 5.2. *Podstawowa hierarchiczna struktura strony HTML, w której znaczniki obejmują inne tagi, jest często przedstawiana w formie drzewa genealogicznego. Znaczniki, które zawierają inne tagi, są nazywane „przodkami", a znaczniki wewnętrzne to „potomkowie"*

Obok samych znaczników HTML przeglądarka śledzi zapisany w nich tekst (na przykład „Nagłówek" w znaczniku `<h1>` na rysunku 5.2) i *atrybuty* przypisane do każdego tagu (takie jak atrybut `class` tagów `<body>` i `<h1>` na tymże rysunku). W modelu DOM wszystkie te jednostki — znaczniki (*elementy*), atrybuty i tekst — tworzą odrębne *węzły*.

Pobieranie elementów strony

Dla przeglądarki strona to po prostu uporządkowany zbiór znaczników, ich atrybutów i tekstu, czyli — w języku specyficznym dla modelu DOM — zbiór *węzłów*. Dlatego aby móc manipulować elementami strony w kodzie JavaScript, niezbędny jest dostęp do węzłów strony. Do ich pobierania służą dwie podstawowe metody: `getElementById()` i `getElementsByTagName()`.

Metoda getElementById()

Przy użyciu tej metody można zlokalizować konkretny węzeł o określonym identyfikatorze. Znacznik `<h1>` na rysunku 5.2 ma atrybut `ID` o wartości `header`. Poniższy kod JavaScript pobiera ten węzeł:

```
document.getElementById('header')
```

W tłumaczeniu na polski wiersz ten oznacza: „Znajdź na stronie znacznik o atrybucie `ID` o wartości `'header'`". Słowo `document` w instrukcji `document.getElementById`

↪`('header')` to słowo kluczowe, które określa cały dokument. Jest ono niezbędne — nie wystarczy użyć członu `getElementById()`. Polecenie `getElementById()` to nazwa metody dokumentu, a `'header'` to łańcuch znaków (nazwa szukanego identyfikatora) przesyłany do tej metody jako argument. Więcej informacji o argumentach znajdziesz na stronie 108.

Uwaga: Metoda `getElementById()` wymaga podania jednego łańcucha znaków — wartości atrybutu `ID` znacznika, na przykład:

```
document.getElementById('header')
```

Jednak nie oznacza to, że do metody trzeba przekazać literał znakowy. Można także użyć zmiennej zawierającej szukany identyfikator:

```
var lookFor = 'header';
var foundNode = document.getElementById(lookFor);
```

Często wynik działania tej metody jest przypisywany do zmiennej, która umożliwia manipulowanie danym znacznikiem w dalszej części programu. Załóżmy, że chcesz użyć kodu JavaScript do zmiany tekstu nagłówka w dokumencie HTML przedstawionym na stronie 159. Można to zrobić w następujący sposób:

```
var headLine = document.getElementById('header');
headLine.innerHTML = 'JavaScript był tutaj!';
```

Polecenie `getElementById()` zwraca referencję do pojedynczego węzła, którą program zapisuje w zmiennej `headLine`. Zapisanie wyniku działania tej metody w zmiennej to bardzo wygodne rozwiązanie. Dzięki niemu przy manipulowaniu znacznikiem można używać nazwy zmiennej, a nie dużo dłuższego zapisu `document.` ↪`getElementById('idName')`. Drugi wiersz kodu używa tej zmiennej do uzyskania dostępu do właściwości `innerHTML` znacznika: `headLine.innerHTML` (omówienie tej właściwości znajdziesz na stronie 165).

Metoda getElementsByTagName()

Czasem pobieranie jednego elementu za pomocą metody `getElementById()` nie jest najwygodniejszym podejściem. Możliwe, że chcesz pobrać wszystkie odnośniki ze strony i wykonać na nich określoną operację, na przykład sprawić, aby otwierały strony spoza witryny w nowym oknie. Potrzebna jest do tego lista elementów, a nie tylko jeden znacznik o określonym identyfikatorze. Polecenie `getElementsByTag` ↪`Name()` umożliwia pobranie takiej listy.

Ta metoda działa podobnie jak polecenie `getElementById()`, jednak nie przyjmuje identyfikatora, a nazwę szukanych znaczników. Na przykład aby znaleźć wszystkie odnośniki zapisane na stronie, można użyć poniższego kodu:

```
var pageLinks = document.getElementsByTagName('a');
```

Ten wiersz oznacza: „Znajdź w dokumencie każdy znacznik `<a>` i zapisz wyniki w zmiennej `pageLinks`". Metoda `getElementsByTagName()` zwraca listę węzłów, a nie pojedynczy element. Dlatego wynikowa zmienna przypomina tablicę. Można pobrać z niej pojedynczy węzeł za pomocą indeksu, sprawdzić łączną liczbę wartości przy użyciu właściwości `length` i przejść w pętli `for` po elementach (zobacz stronę 102).

Na przykład pierwszym elementem zapisanym w zmiennej `pageLinks` (`page ↪Links[0]`) będzie pierwszy znacznik `<a>` na stronie, a właściwość `pageLinks. ↪length` zwróci liczbę wszystkich takich tagów.

Wskazówka: Łatwo popełnić literówkę przy korzystaniu z tych metod. Najczęstszym błędem początkujących (a także doświadczonych) programistów jest zapisanie członu `Id` za pomocą dwóch dużych liter. Ponadto trzeba pamiętać o literze `s` w słowie `Elements` (to liczba mnoga) w nazwie metody `getElementsByTagName()`:

```
document.getElementById('banner');
document.getElementsByTagName('a');
```

Metod `getElementById()` i `getElementsByTagName()` można też używać wspólnie. Załóżmy, że na stronie znajduje się znacznik `<div>` o identyfikatorze `'banner'`. Aby sprawdzić, ile odnośników zawiera ten znacznik, należy użyć metody `get ↪ElementById()` w celu pobrania tego elementu, a następnie sprawdzić jego zawartość za pomocą metody `getElementsByTagName()`:

```
var banner = document.getElementById('banner');
var bannerLinks = banner.getElementsByTagName('a');
var totalBannerLinks = bannerLinks.length;
```

Choć wyszukiwanie elementów za pomocą identyfikatora służy do przeszukiwania dokumentu (`document.getElementById()`), znaczników określonego typu można szukać zarówno w całym dokumencie (`document.getElementsByTagName()`), jak i w poszczególnych węzłach. W przedstawionym wcześniej kodzie zmienna `banner` zawiera referencję do znacznika `<div>`, dlatego instrukcja `banner.getElements ↪ByTagName('a')` wyszukuje tagi `<a>` tylko *wewnątrz* danego elementu `<div>`.

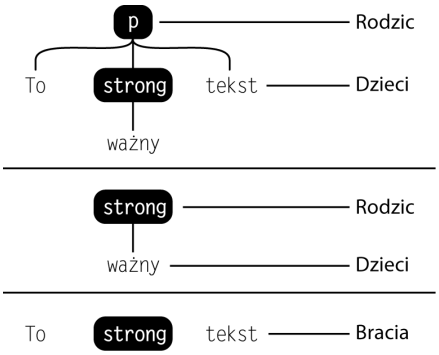
Pobieranie pobliskich węzłów

Także tekst jest uznawany za węzeł, dlatego napisowi „Nagłówek" w znaczniku `<h1>` i samemu znacznikowi odpowiadają odrębne węzły (zobacz stronę 159). Oznacza to, że jeśli pobierzesz tag `<h1>` za pomocą opisanych wcześniej technik, uzyskasz dostęp tylko do znacznika, a nie do zapisanego w nim tekstu. Jak więc można pobrać zawartość tagu? Niestety, jedyne rozwiązanie udostępniane przez model DOM wymaga użycia pewnej sztuczki. Trzeba zacząć od węzła `<h1>`, przejść do węzła tekstowego, a następnie pobrać jego wartość.

Aby zrozumieć ten proces, trzeba zapoznać się z relacjami między znacznikami. Jeśli używałeś już stylów CSS, prawdopodobnie znasz selektory *potomków*. Są one jednym z najwartościowszych narzędzi arkuszy CSS, ponieważ umożliwiają formatowanie znaczników na podstawie ich powiązań z innymi tagami. Dzięki temu można sprawić, że akapity (znaczniki `<p>`) w ramce bocznej strony będą wyglądały inaczej niż akapity umieszczone w stopce.

W selektorach podrzędnych uwzględniane są relacje opisane na rysunku 5.2. Jeśli znacznik znajduje się wewnątrz innego tagu, jest jego potomkiem. Znacznik `<h1>` w przykładowym kodzie HTML (zobacz stronę 159) jest potomkiem tagu `<body>`, a także — pośrednio — tagu `<html>`. Znaczniki zawierające inne tagi są nazywane *przodkami*. Na rysunku 5.2 znacznik `<p>` to przodek tagu `<strong>`.

W modelu DOM relacje występują także między innymi znacznikami, jednak dostęp jest ograniczony do „najbliższej rodziny". Oznacza to, że można dotrzeć do węzła „rodzica", „dziecka" i „brata". Rysunek 5.3 ilustruje te relacje. Jeśli węzeł znajduje się bezpośrednio w innym węźle, tak jak tekst „To" w znaczniku `<p>`, jest *dzieckiem*. Węzeł bezpośrednio zawierający inny węzeł, tak jak znacznik `<strong>`, który zawiera tekst „ważny", jest *rodzicem*. Węzły mające tego samego rodzica, na przykład dwa węzły tekstowe, „To" i „tekst", oraz znacznik `<strong>`, to *bracia*.



Rysunek 5.3. *W obecnym standardzie modelu DOM nie ma kuzynów pierwszego stopnia, babci ciotecznych ani nawet dziadków. Uwzględniane relacje to: rodzic, dziecko i brat*

Model DOM udostępnia kilka metod dostępu do pobliskich węzłów. Są to:

- Właściwość `.childNodes` węzła. Zawiera ona tablicę wszystkich dzieci danego węzła. Ta lista działa podobnie jak tablica zwracana przez metodę `getElements➥ByTagName()` (zobacz stronę 161). Załóżmy, że programista dodał do pliku HTML ze strony 159 następujący kod JavaScript:

  ```
  var headline = document.getElementById('header');
  var headlineKids = headline.childNodes;
  ```

  Zmienna `headlineKids` będzie zawierać listę wszystkich dzieci znacznika o identyfikatorze `'headline'` (czyli tagu `<h1>`). Tu istnieje tylko jeden taki element — węzeł tekstowy o wartości „Nagłówek". Dlatego jeśli chcesz sprawdzić, jaki tekst zawiera ten węzeł, możesz użyć następującego wiersza kodu:

  ```
  var headlineText = headlineKids[0].nodeValue;
  ```

  Instrukcja `headlineKids[0]` zapewnia dostęp do pierwszego dziecka w tablicy. Ponieważ jest to jedyne dziecko elementu `<h1>` (rysunek 5.2), jest też jedynym węzłem na liście. Aby pobrać tekst tego węzła, należy użyć właściwości `node➥Value`. Podobny efekt można uzyskać także w łatwiejszy sposób, który poznasz na stronie 164.

- Właściwość `.parentNode` reprezentuje rodzica danego węzła. Jeśli chcesz sprawdzić, w jakim węźle znajduje się znacznik `<h1>` z rysunku 5.2, możesz użyć poniższego kodu:

  ```
  var headline = document.getElementById('header');
  var headlineParent = headline.parentNode;
  ```

  Zmienna `headlineParent` będzie zawierać referencję do znacznika `<body>`.

- Właściwości `.nextSibling` i `.previousSibling` wskazują na węzeł znajdujący się bezpośrednio przed bieżącym elementem lub po nim. Na rysunku 5.2 znaczniki `<h1>` i `<p>` są braćmi (znacznik `<p>` pojawia się bezpośrednio po zamykającym znaczniku `</h1>`).

  ```
  var headline = document.getElementById('header');
  var headlineSibling = headline.nextSibling;
  ```

  Zmienna `headlineSibling` to referencja do znacznika `<p>`, który znajduje się po tagu `<h1>`. Jeśli spróbujesz uzyskać dostęp do nieistniejącego brata, JavaScript zwróci wartość `null` (zobacz wskazówkę na stronie 135). Do sprawdzenia, czy węzeł ma wcześniejszego brata (właściwość `.previousSibling`), można użyć następującego kodu:

  ```
  var headline = document.getElementById('header');
  var headlineSibling = headline.previousSibling;
  if (! headlineSibling) {
    alert('Ten węzeł nie ma młodszego brata');
  } else {
    // Przeprowadzanie operacji na węźle-bracie.
  }
  ```

Poruszanie się po strukturze DOM strony wymaga sporo zachodu. Na przykład aby pobrać cały tekst znacznika `<p>` widocznego na rysunku 5.2, trzeba uzyskać dostęp do listy jego dzieci i zapisać tekst każdego z nich. Po napotkaniu znacznika `<strong>` (rysunek 5.2) trzeba ponownie pobrać dziecko! Na szczęście na stronie 170 poznasz dużo łatwiejszy sposób korzystania z modelu DOM.

Dodawanie zawartości do strony

Programy JavaScript często muszą dodawać, usuwać i zmieniać zawartość strony. W quizie napisanym w rozdziale 3. (strona 113) użyłeś metody `document.write()`, aby dodać do strony ostateczny wynik gracza. W witrynie Netflix (rysunek 5.1) umieszczenie kursora nad plakatem do filmu powoduje wyświetlenie na stronie jego opisu.

Uwaga: W poprzednich rozdziałach do wyświetlania na stronie tekstu wygenerowanego w kodzie JavaScript używałeś polecenia `document.write()` (zobacz na przykład stronę 40). Ta instrukcja jest łatwa do zrozumienia i w użytkowaniu, jednak ma bardzo ograniczone możliwości. Umożliwia dodawanie nowej treści, jednak nie pozwala między innymi na modyfikowanie istniejącego kodu. Ponadto polecenie to jest uruchamiane w czasie wczytywania strony, dlatego nie można dodać tekstu później, na przykład po kliknięciu przycisku albo wpisaniu danych w polu formularza.

Dodawanie treści za pomocą modelu DOM to duże wyzwanie. Proces ten wymaga utworzenia potrzebnych węzłów, a następnie dołączenia ich do strony. Jeśli chcesz wstawić znacznik `<div>`, zawierający kilka innych tagów i tekst, musisz utworzyć zbiór węzłów i umieścić je na stronie z uwzględnieniem powiązań między nimi. Na szczęście producenci przeglądarek udostępniają dużo prostsze rozwiązanie — właściwość `innerHTML`.

Ta właściwość nie jest standardową częścią modelu DOM. Po raz pierwszy wprowadzono ją w Internet Explorerze, jednak obecnie udostępniają ją wszystkie współczesne przeglądarki z obsługą języka JavaScript. Właściwość `innerHTML` reprezentuje

cały kod HTML węzła. Spójrz na kod HTML przedstawiony na stronie 159. Znacznik <p> obejmuje fragment kodu HTML, a jego właściwość innerHTML ma wartość To ważny tekst. Dostęp do tego fragmentu w kodzie JavaScript można uzyskać w następujący sposób:

```
// Pobieranie listy wszystkich znaczników <p> na stronie.
var pTags = document.getElementsByTagName('p');
// Pobieranie pierwszego znacznika <p> na stronie.
var theP = pTags[0];
alert(theP.innerHTML);
```

Zmienna theP reprezentuje węzeł pierwszego akapitu strony. Ostatni wiersz wyświetla okno dialogowe z kodem tego akapitu. Po uruchomieniu powyższego fragmentu kodu JavaScript w dokumencie HTML przedstawionym na stronie 159 pojawi się okienko z napisem „To ważny tekst".

Uwaga: Właściwość innerHTML może zostać włączona do specyfikacji języka HTML 5, rozwijanego przez organizację W3C (zobacz stronę *www.w3.org/TR/html5*).

Za pomocą właściwości innerHMTL można nie tylko sprawdzić zawartość węzła, ale także ją zmodyfikować:

```
var headLine = document.getElementById('header');
headLine.innerHTML = 'JavaScript był tutaj!';
```

Ten kod zmienia zawartość znacznika o identyfikatorze 'header' na "JavaScript był tutaj!". Oprócz tekstu można dodawać także inne elementy, na przykład całe fragmenty kodu HTML, w tym znaczniki i ich atrybuty. Przykład zastosowania tego podejścia znajdziesz w następnym punkcie.

Księżycowy quiz — wersja druga

W rozdziale 3. utworzyłeś program JavaScript, który zadawał pytania za pomocą polecenia prompt() i wyświetlał wyniki przy użyciu polecenia document.write(). W tym krótkim przykładzie zmodyfikujesz tamten skrypt, wykorzystując poznane w tym rozdziale techniki oparte na modelu DOM.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 38.

1. **Otwórz w edytorze tekstu plik *5.1.html* z rozdziału *R05*.**

   Ten plik zawiera kompletną wersję przykładu 3.3 ze strony 118.

2. **Znajdź poniższy kod HTML i usuń kod JavaScript wyróżniony pogrubieniem:**

```
<p>
  <script type="text/javascript">
    var message = 'Liczba punktów: ' + score;
    message += ' z ' + questions.length;
    message += '.';
    document.write(message);
  </script>
</p>
```

Na stronie powinien pozostać pusty akapit. Posłuży on do wyświetlania wyników quizu. Aby ułatwić dostęp do tego akapitu, warto dodać do niego identyfikator.

3. **Dodaj atrybut `id="quizResults"` do znacznika `<p>`. Kod HTML powinien wyglądać teraz następująco:**

```
<h1>Prosty quiz – wersja druga</h1>
<p id="quizResults"></p>
</div>
```

Następnie trzeba utworzyć funkcję, która przechodzi przez listę pytań, a potem wyświetla wyniki na stronie.

4. **Znajdź pętlę `for` w bloku kodu JavaScript w górnej części strony. Umieść wokół pętli kod wyróżniony pogrubieniem:**

```
function doQuiz() {
  // Przechodzi przez listę pytań i zadaje je.
  for (var i=0; i<questions.length; i++) {
    askQuestion(questions[i]);
  }
}
```

Nie zapomnij zamknąć nawiasu klamrowego w ostatnim wierszu. Znak ten kończy nową funkcję, zawierająca pętlę `for`, która przechodzi po wszystkich elementach tablicy `questions`. Zadania wykonywane przez tę funkcję poznasz już za chwilę.

Następnie należy dodać informację o wyniku gracza. Do jej wyświetlenia posłuży ten sam kod, którego użyłeś w pliku *3.3.html*.

5. **Między zamykającym nawiasem klamrowym pętli a końcowym nawiasem klamrowym funkcji dodaj trzy wiersze kodu JavaScript:**

```
var message = 'Liczba punktów: ' + score;
message += ' z ' + questions.length;
message += '.';
```

Ten kod pochodzi z przykładu 3.3, dlatego jeśli nie chcesz go przepisywać, możesz go skopiować i wkleić. Na tym etapie skrypt nie różni się zbytnio od przykładu 3.3. Program wyświetla pytania, a następnie wynik. Teraz należy wykorzystać model DOM. Najpierw dodaj referencję do pustego znacznika `<p>`.

6. **Wciśnij klawisz *Enter*, aby dodać nowy, pusty wiersz pod trzema dodanymi wcześniej instrukcjami. Następnie wpisz następujący kod:**

```
var resultArea = document.getElementById('quizResults');
```

Ten wiersz wyszukuje w dokumencie znacznik o dodanym w kroku 3. identyfikatorze `'quizResults'`, a następnie zapisuje referencję do tego tagu w zmiennej `resultArea`. Do tej referencji można przypisać informację o wyniku.

7. **Ponownie wciśnij klawisz *Enter* i wpisz fragment `resultArea.innerHTML = message;`. Kompletny kod funkcji powinien wyglądać następująco:**

```
function doQuiz() {
  // Przechodzi przez listę pytań i zadaje je.
  for (var i=0; i<questions.length; i++) {
    askQuestion(questions[i]);
  }
  var message = 'Liczba punktów: ' + score;
  message += ' z ' + questions.length;
  message += '.';
```

```
    var resultArea = document.getElementById('quizResults');
    resultArea.innerHTML = message;
}
```

Ostatni wiersz funkcji przypisuje wartość zmiennej `message` do właściwości `innerHTML` znacznika `<p>`, czyli zapisuje informację o wyniku w akapicie (podobnie jak wcześniej polecenie `document.write()`). Podejście oparte na właściwości `innerHTML` jest prostsze, ponieważ nie wymaga dodawania drugiego bloku kodu JavaScript w ciele strony, co jest konieczne przy używaniu polecenia `document.write()`.

Pamiętaj, że funkcje są uruchamiane dopiero po ich wywołaniu (zobacz stronę 107). Dlatego utworzenie funkcji, która zadaje pytania i wyświetla wynik, to nie wszystko. Program przeprowadzi quiz dopiero po wywołaniu funkcji `doQuiz()`. Dlatego trzeba dodać kod, który to zrobi.

8. **Znajdź końcowy znacznik `</script>` (pod całym kodem JavaScript w sekcji `<head>`) i dodaj przed nim wyróżnione pogrubieniem wywołanie:**

```
window.onload=doQuiz;
</script>
```

Witaj we wspaniałym świecie zdarzeń języka JavaScript. Dodany wiersz kodu nakazuje interpreterowi uruchomienie funkcji `doQuiz()` *po* wczytaniu strony. Fragment `onload` to tak zwany **uchwyt** *zdarzenia*. Zdarzenie to moment wystąpienia określonego zjawiska w przeglądarce lub na stronie. Kiedy przeglądarka wczyta stronę, zachodzi zdarzenie `load`. Kiedy użytkownik umieści kursor nad odnośnikiem, ma miejsce zdarzenie `mouseover`. Uchwyty zdarzeń umożliwiają przypisanie funkcji do zdarzenia, czyli określenie, co przeglądarka powinna zrobić, kiedy zajdzie dane zdarzenie. Więcej o zdarzeniach dowiesz się w następnym rozdziale.

Dlaczego nie można uruchomić quizu przed wczytaniem strony? W końcu tak właśnie działał przykład 3.3 ze strony 113. Jeśli otworzysz w przeglądarce gotowy plik *3.3.html* (lub udostępniony dokument *kompletny_3.3.html* z katalogu *R03*), zauważysz, że strona jest zupełnie pusta w czasie zadawania pytań (górna część rysunku 5.4). Dzieje się tak, ponieważ kod JavaScript quizu jest uruchamiany bezpośrednio po napotkaniu pętli `for` przez interpreter. Nie czeka on na wczytanie i wyświetlenie kodu HTML, dlatego przeglądarka może wygenerować stronę dopiero po zadaniu wszystkich pytań.

Zapisz ukończony wcześniej plik *5.1.html* i wyświetl go w przeglądarce. Tym razem przeglądarka wyświetli stronę przed zadaniem pytań (dolna część rysunku 5.4). Jest to efekt użycia instrukcji `window.onload=doQuiz`, która stanowi dla interpretera informację nakazującą uruchomienie quizu dopiero po wczytaniu i wyświetleniu strony. To rozwiązanie nie tylko wygląda dużo lepiej (pusta strona może rozpraszać uwagę), ale jest niezbędne, jeśli chcesz używać modelu DOM do manipulowania zawartością strony.

Kod JavaScript na omawianej stronie znajduje się *przed* kodem HTML. W czasie wczytywania strony przeglądarka nie ma informacji o fragmentach HTML z końcowej części strony. Dlatego akapit, w którym skrypt ma wyświetlać wynik („Liczba punktów: 3 z 3."), dla przeglądarki jeszcze nie istnieje. Jeśli spróbujesz uruchomić

Rysunek 5.4. *Kiedy uruchomisz program JavaScript przed wczytaniem strony, nie będzie ona widoczna do momentu zakończenia działania skryptu. Na stronie widocznej w górnej części rysunku kod JavaScript musi zakończyć zadawanie pytań, zanim przeglądarka będzie mogła wyświetlić stronę. Jeśli jednak użyjesz zdarzenia onload, przeglądarka wczyta i wyświetli stronę, a dopiero wtedy uruchomi skrypt (dolna część rysunku)*

nowy kod JavaScript *natychmiast* (przed wczytaniem kodu HTML), przeglądarka poinformuje o błędzie w momencie, kiedy skrypt będzie chciał pobrać znacznik <p> i zapisać w nim komunikat. Problem wynika z tego, że interpreter nie ma informacji o tym znaczniku.

Dlatego w kroku 4. umieściłeś w funkcji pętlę, kod generujący komunikat, a także kod, który pobiera znacznik `<p>` i zapisuje w nim informacje. Dzięki tej funkcji przeglądarka może wczytać stronę i umieścić ją w pamięci, a dopiero *potem* wykonać wszystkie operacji związane z quizem.

Możesz się zastanawiać, dlaczego po funkcji w wierszu `window.onload=doQuiz` nie ma nawiasów. Na stronie 107 dowiedziałeś się, że w wywołaniach funkcji zawsze należy ich używać (na przykład `doQuiz()`). Nawiasy powodują, że funkcja jest uruchamiana *natychmiast*. Dlatego wiersz `window.onload=doQuiz()` spowoduje natychmiastowe przeprowadzenie quizu, jeszcze przed wczytaniem strony. Jednak instrukcja `window.onload=doQuiz` tylko określa funkcję, bez jej wywoływania. Skrypt uruchomi tę funkcję dopiero po wczytaniu strony. Skomplikowane? To prawda, jednak właśnie tak działa język JavaScript. Więcej informacji o tej technice znajdziesz na stronie 217.

Wady modelu DOM

Model DOM to wartościowe narzędzie dla programistów używających języka JavaScript, jednak ma też kilka wad. Na stronie 162 zobaczyłeś, że poruszanie się między węzłami modelu DOM jest czasochłonne. Ponadto model DOM udostępnia tylko kilka sposobów uzyskania dostępu do znaczników — za pomocą identyfikatorów i nazw tagów. Nie można w wygodny sposób znaleźć na przykład wszystkich znaczników określonej klasy, co jest przydatne przy manipulowaniu zbiorem powiązanych elementów, takich jak rysunki o klasie `slideshow` używane w pokazie slajdów opartym na języku JavaScript.

Następnym utrudnieniem są różnice w obsłudze modelu DOM przez poszczególne przeglądarki. Techniki opisane na poprzednich stronach działają we wszystkich przeglądarkach, jednak niektóre elementy standardu DOM sprawiają problemy. Przeglądarka Internet Explorer obsługuje zdarzenia inaczej niż pozostałe przeglądarki, ten sam kod HTML może dać inną liczbę węzłów w przeglądarkach Firefox i Safari niż w Internet Explorerze, a ponadto Internet Explorer nie zawsze pobiera atrybuty znaczników HTML w taki sam sposób, jak robią to Firefox, Safari i Opera. Ponadto poszczególne przeglądarki w odmienny sposób traktują odstępy (na przykład tabulacje i spacje) w kodzie HTML — niektóre przy ich napotkaniu tworzą nowe węzły (Firefox i Safari), a Internet Explorer ignoruje takie znaki. A to tylko kilka różnic w obsłudze modelu DOM przez najpopularniejsze przeglądarki!

Przezwyciężenie takich problemów w kodzie JavaScript jest tak poważnym zagadnieniem, że można poświęcić mu całą (bardzo nudną) książkę. W wielu pozycjach dotyczących tego języka znajdują się długie opisy kodu potrzebnego do zapewnienia prawidłowego działania stron w różnych przeglądarkach. Jednak życie jest zbyt krótkie, aby zaprzątać sobie głowę takimi problemami. Lepiej zająć się tworzeniem interaktywnych interfejsów użytkownika i dodawaniem ciekawych efektów do witryn, zamiast zastanawiać się, co zrobić, aby kod działał tak samo w Internet Explorerze, Firefoksie, Safari i Operze. Dlatego w tej książce pominięto wiele przytłaczających szczegółów potrzebnych do zapewnienia działania podstawowych funkcji DOM w różnych przeglądarkach. W zamian wykorzystasz bardzo zaawansowany,

bezpłatny kod JavaScript, którego możesz użyć do szybkiego budowania opartych na tym języku stron wyświetlanych prawidłowo w każdej przeglądarce. W następnym punkcie dowiesz się, jak pobrać taki kod.

Biblioteki języka JavaScript

Wiele skryptów JavaScriptu wykonuje te same operacje, takie jak pobieranie elementów, dodawanie nowej zawartości, ukrywanie i wyświetlanie treści, modyfikowanie atrybutów znaczników, określanie wartości pól formularza lub określanie reakcji programu na różne działania użytkowników. Szczegóły obsługi tych operacji mogą być dość skomplikowane, zwłaszcza jeśli chcesz, aby program działał prawidłowo we wszystkich popularnych przeglądarkach. Na szczęście *biblioteki* języka JavaScript pozwalają pominąć wiele czasochłonnych drobiazgów programistycznych.

Biblioteki JavaScript to kolekcje kodu JavaScript, które udostępniają proste rozwiązania wielu żmudnych, codziennych problemów. Możesz traktować biblioteki jak zbiory funkcji języka JavaScript, które można dodawać do własnych stron. Funkcje te ułatwiają wykonywanie standardowych zadań i często pozwalają zastąpić jednym wywołaniem wiele wierszy własnego kodu JavaScript (oraz zaoszczędzić długich godzin potrzebnych na testy). Oznacza to, że liczne fragmenty programu ktoś już napisał za Ciebie! Dostępnych jest wiele bibliotek JavaScript, a niektóre z nich są podstawą popularnych witryn, takich jak Yahoo!, NBC, Amazon, Digg, CNN, Apple, Microsoft i Twitter.

W tej książce użyto popularnej biblioteki jQuery (*www.jquery.com*). Dostępne są też inne biblioteki (zobacz ramkę na stronie 171), jednak jQuery ma wiele zalet:

- **Stosunkowo mały rozmiar pliku**. Zminimalizowana wersja biblioteki zajmuje tylko około 55 kilobajtów, a w pełni skompresowany pakiet ma tylko 30 kilobajtów.
- **Jest łatwa do zrozumienia dla projektantów stron WWW**. Używanie jQuery nie wymaga zaawansowanej wiedzy programistycznej. Potrzebna jest tylko znajomość stylów CSS, które nie są niczym nowym dla większości projektantów stron WWW.
- **Jest dobrze przetestowana**. Biblioteka jQuery jest używana na tysiącach stron, w tym w wielu popularnych, często odwiedzanych witrynach, takich jak Digg, Dell, the Onion, Warner Bros, Records, NBC i Newsweek. Nawet firma Google używa tej biblioteki. Popularność jQuery to dowód jej jakości.
- **Jest bezpłatna**. Tej oferty nikt nie przebije!
- **Jest rozwijana przez dużą społeczność programistów**. Kiedy czytasz tę książkę, nad projektem jQuery pracuje wiele osób, które piszą kod, naprawiają błędy, dodają nowe funkcje oraz aktualizują dokumentację i samouczki w witrynie poświęconej tej bibliotece. Biblioteka JavaScript utworzona przez jednego programistę może szybko zniknąć, jeśli autor straci zainteresowanie projektem, natomiast dzięki wysiłkom programistów z całego świata jQuery prawdopodobnie będzie dostępna przez długi czas. W pewnym sensie wielu programistów języka JavaScript pracuje dla Ciebie za darmo.

- **Wtyczki**. Biblioteka jQuery umożliwia innym programistom tworzenie *wtyczek* — dodatkowych programów napisanych w języku JavaScript, które współdziałają z tą biblioteką. Niezwykle ułatwia to dodawanie obsługi zadań, efektów i funkcji do stron WWW. W tej książce poznasz wtyczki, które sprawiają, że walidację formularzy, rozwijane menu nawigacyjne lub interaktywny pokaz slajdów można wbudować w witrynę w pół godziny zamiast w dwa tygodnie. Dostępne są dosłownie setki innych wtyczek powiązanych z tą biblioteką.

W tej książce użyłeś już biblioteki jQuery. W przykładzie w rozdziale 1. (strona 41) dodałeś kilka wierszy kodu JavaScript, aby szybko i łatwo zmienić kolor co drugiego wiersza tabeli.

WIEDZA W PIGUŁCE

Inne biblioteki

JQuery nie jest jedyną dostępną biblioteką JavaScript. Niektóre z innych bibliotek obsługują specyficzne operacje, a część jest przeznaczona do użytku ogólnego i rozwiązuje niemal każde zadanie związane z językiem JavaScript. Oto lista kilku najbardziej popularnych produktów tego typu:

Yahoo User Interface Library (*http://developer.yahoo.com/yui/*) to projekt firmy Yahoo!, używany w wielu miejscach jej witryny. Programiści z Yahoo! wciąż wzbogacają i usprawniają tę bibliotekę, a w poświęconej jej witrynie można znaleźć świetną dokumentację.

Prototype (*http://www.prototypejs.org/*) to jedna z pierwszych bibliotek JavaScript. Zajmuje 124 kilobajty i umożliwia wykonywanie różnorodnych zadań — od pobierania elementów, co ułatwia manipulowanie modelem DOM, po upraszczanie ajaksowej komunikacji z serwerem. Programiści często używają jej wraz z biblioteką efektów graficznych *scriptaculous* (*http://script.aculo.us/*), która udostępnia animacje i inne ciekawe elementy interfejsu użytkownika.

Dojo Toolkit (*http://dojotoolkit.org/*) to następna biblioteka dostępna od długiego czasu. Jest bardzo rozbudowana i zawiera wiele plików JavaScript, które rozwiązują niemal każde zadanie związane z językiem JavaScript.

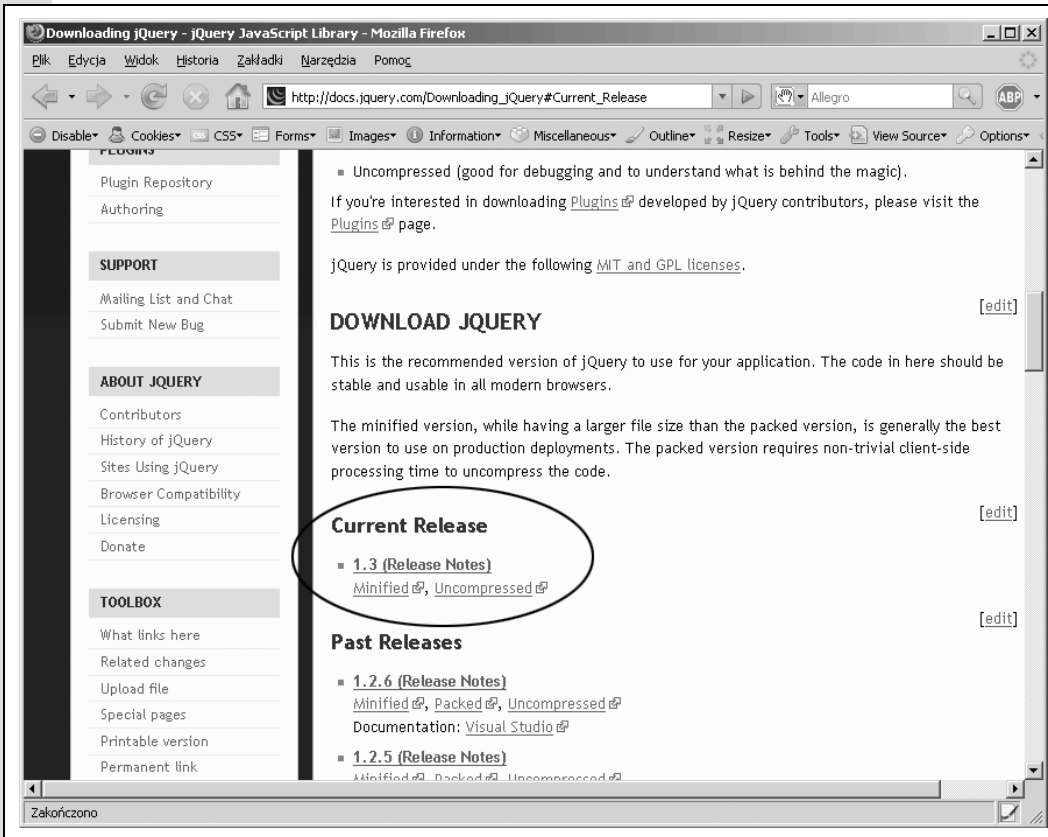
Mootools (*http://mootools.net/*) to kolejna popularna biblioteka z dobrą dokumentacją i świetną witryną WWW.

Wprowadzenie do biblioteki jQuery

Aby rozpocząć korzystanie z jQuery, trzeba ją najpierw pobrać. Biblioteka ta znajduje się w jednym pliku JavaScript, *jquery.js*, który trzeba dołączyć do strony. Przykładowe pliki, które możesz pobrać ze strony poświęconej książce *ftp://ftp.helion.pl/przyklad/jascnp.zip*, obejmują tę bibliotekę, jednak ponieważ rozwijający ją zespół wciąż wprowadza aktualizacje, warto poszukać najnowszej wersji w sekcji *Current Release* (zakreślona na rysunku 5.5) na stronie *http://docs.jquery.com/Downloading_jQuery*.

Plik jQuery można pobrać w trzech wersjach. Wybór jednej z nich zależy od tego, w jaki sposób chcesz używać biblioteki. Oto te wersje:

- **Wersja nieskompresowana**. Nieskompresowany plik jQuery zajmuje najwięcej miejsca (w wersji jQuery 1.2.6 jest to 97,8 kilobajta). Wersji tej nie należy używać we własnej witrynie, ale pomaga ona zrozumieć, jak zbudowana jest ta biblioteka. Kod obejmuje liczne komentarze (zobacz stronę 80), które pozwalają poznać przeznaczenie poszczególnych części pliku. Jednak aby zrozumieć te komentarze, trzeba mieć **dużą** wiedzę o języku JavaScript.



Rysunek 5.5. *Zewnętrzny plik JavaScript biblioteki jQuery ma trzy wersje. Warto wybrać wersję zminimalizowaną (Minified), ponieważ zapewnia najlepszy kompromis między rozmiarem pliku a wydajnością*

- **Wersja skompresowana**. Spakowana biblioteka jQuery zajmuje najmniej miejsca (w wersji jQuery 1.2.6 jest to tylko 30,3 kilobajta). Udostępnia te same funkcje co wersja nieskompresowana, jednak kod JavaScript jest przetworzony przez zaawansowany program kompresujący (*http://dean.edwards.name/packer/*), który zmniejsza liczbę potrzebnych znaków. Wadą tej wersji jest to, że przeglądarka musi „rozpakować" plik przy każdym jego uruchomieniu, co nieco spowalnia działanie strony w porównaniu z wersją nieskompresowaną.
- **Wersja zminimalizowana**. W zminimalizowanym pliku jQuery użyto prostszej metody kompresji niż w pliku *skompresowanym*, dlatego jest on nieco większy (w wersji jQuery 1.2.6 zajmuje 54,5 kilobajta). Jednak ponieważ wersji zminimalizowanej nie trzeba rozpakowywać przy każdym uruchomieniu, po pobraniu działa ona nieco szybciej niż plik skompresowany. Ponadto przeglądarki zwykle zapisują pobrany plik jQuery w pamięci podręcznej, dlatego jego rozmiar nie jest najważniejszy. Przeglądarka musi pobrać bibliotekę tylko raz, a kiedy użytkownik przejdzie do innej strony witryny, można użyć dostępnego już pliku. Ponieważ wersja zminimalizowana jest dość mała i działa szybko, właśnie jej będziesz używał w przykładach w tej książce.

Po pobraniu pliku jQuery umieść go na przykład w katalogu głównym witryny. Niektórzy projektanci stron WWW tworzą odrębny folder na pliki JavaScript (jego popularne nazwy to *js* i *libs*) i zapisują w nim bibliotekę jQuery, a także inne pliki *.js*.

Wskazówka: Nazwa pliku jQuery z witryny *jQuery.com* zawiera informacje o numerze wersji i typie kompresji. Na przykład *jquery-1.2.6.min.js* to plik zminimalizowanej wersji *jQuery 1.2.6*. Możesz zmienić tę nazwę na prostszą, na przykład *jquery126.js* lub *jquery.js*.

Aby użyć pobranego pliku, trzeba dołączyć go do strony. Jest to zwykły zewnętrzny plik *.js*, dlatego można dodać go w standardowy sposób, opisany na stronie 35. Załóżmy, że zapisałeś plik *jquery.js* w podkatalogu *js* katalogu głównego witryny. Aby dołączyć ten plik do strony głównej, umieść w sekcji nagłówkowej poniższy znacznik `<script>`:

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

Po dołączeniu pliku jQuery można dodać własne skrypty, które korzystają z zaawansowanych funkcji tej biblioteki. Możesz na przykład dołączyć zewnętrzny plik JavaScript z własnym kodem lub wpisać instrukcję w drugim znaczniku `<script>`:

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
  // Tu kod skryptu.
</script>
```

Pobieranie elementów strony — podejście drugie

Na stronie 160 dowiedziałeś się, że model DOM udostępnia dwie podstawowe metody pobierania elementów stron WWW — `document.getElementById()` i `document.getElementsByTagName()`. Niestety, te dwie metody mają pewne ograniczenia. Na przykład jeśli zechcesz pobrać wszystkie znaczniki `<a>` klasy `navButton`, będziesz musiał najpierw uzyskać dostęp do każdego tagu, a następnie sprawdzić, czy ma odpowiednią klasę. Podobny problem występuje między innymi przy pobieraniu co drugiego wiersza tabeli, co było potrzebne w przykładzie z rozdziału 1.

Na szczęście biblioteka jQuery udostępnia bardzo zaawansowaną technikę pobierania i używania kolekcji elementów. W metodzie tej wykorzystano selektory CSS. Jeśli używasz arkuszy CSS do nadawania stylu witrynom, jesteś gotów do korzystania z biblioteki jQuery. Selektor CSS to instrukcja informująca przeglądarkę, do których znaczników należy zastosować styl. Na przykład `h1` to prosty selektor elementów, który pozwala dołączyć styl do wszystkich znaczników `<h1>`, a selektor klasy `.copyright` umożliwia nadanie stylu wszystkich tagom, których atrybut `class` ma wartość `copyright`:

```
<p class="copyright">Copyright, 2009</p>
```

Korzystając z biblioteki jQuery, można pobierać elementy przy użyciu specjalnego polecenia, nazywanego *obiektem jQuery*. Podstawowa składnia takich poleceń wygląda następująco:

```
$('selektor')
```

Przy tworzeniu obiektów jQuery można używać prawie wszystkich selektorów języka CSS 2.1 i wielu selektorów języka CSS 3, nawet jeśli przeglądarka nie obsługuje danego selektora (na przykład Internet Explorer nie rozpoznaje niektórych selektorów języka CSS 3). Jeśli chcesz pobrać za pomocą biblioteki jQuery znacznik o identyfikatorze `banner`, możesz użyć następującego kodu:

```
$('#banner')
```

Łańcuch `#banner` to selektor CSS używany do nadawania stylu znacznikowi o identyfikatorze `banner`. Znak # określa, że programista podaje identyfikator. Po pobraniu elementów należy ich oczywiście użyć. Biblioteka jQuery udostępnia wiele narzędzi do manipulowania znacznikami. Załóżmy, że chcesz zmienić kod HTML elementu. Możesz to zrobić w następujący sposób:

```
$('#banner').html('<h1>JavaScript był tutaj</h1>');
```

Omówienie manipulowania znacznikami za pomocą biblioteki jQuery zaczyna się na stronie 182 i ciągnie do końca książki. Najpierw jednak powinieneś nauczyć się pobierać elementy za pomocą tej biblioteki.

Podstawowe selektory

Podstawowe *selektory* CSS, na przykład selektory identyfikatorów, klas i elementów, to kluczowa część języka CSS. Selektory to doskonały mechanizm do pobierania wielu znaczników za pomocą biblioteki jQuery.

Ponieważ czytanie o selektorach nie jest najlepszym sposobem na ich zrozumienie, wśród przykładowych plików znajdziesz interaktywną stronę WWW, na której będziesz mógł przetestować ten mechanizm. W katalogu *R05* poszukaj pliku *selectors.html*, a następnie otwórz go w przeglądarce. Teraz możesz przetestować różne selektory obsługiwane przez bibliotekę jQuery przez wpisanie ich w odpowiednim polu i kliknięcie przycisku *Zastosuj* (rysunek 5.6).

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 38.

Selektory identyfikatorów

Przy użyciu jQuery i selektora `ID` języka CSS możesz pobrać dowolny element, który ma przypisany identyfikator. Załóżmy, że na stronie znajduje się poniższy fragment kodu HTML:

```
<p id="message">Komunikat specjalny</p>
```

Aby pobrać ten element przy użyciu modelu DOM, możesz użyć następującego kodu:

```
var messagePara = document.getElementById('message');
```

Metoda oparta na jQuery wygląda tak:

```
var messagePara = $('#message');
```

Inaczej niż w metodzie związanej z modelem DOM, nie wystarczy podać nazwy identyfikatora (`'message'`). Należy użyć składni języka CSS — `'#message'`. Trzeba więc dodać znak kratki przez nazwą identyfikatora, podobnie jak przy tworzeniu stylu CSS dla elementu o danym identyfikatorze.

Rysunek 5.6. *Strona selectors.html, dostępna wśród przykładowych plików, umożliwia przetestowanie selektorów biblioteki jQuery. Wpisz wybrany z nich w polu Selektor (oznaczone owalem), a następnie kliknij przycisk Zastosuj. Strona przekształci selektor na obiekt jQuery, a następnie wyświetli pasujące elementy przy użyciu czcionki w kolorze czerwonym. Pod polem Selektor znajduje się kod potrzebny bibliotece do pobrania elementu, a także liczba znalezionych znaczników. Na rysunku selektorem jest h2, a wszystkie pięć znaczników <h2> na stronie zostało wyróżnionych kolorem czerwonym (który wygląda tu zaskakująco podobnie do szarego)*

Selektor elementów

JQuery udostępnia zastępnik dla metody `getElementsByTagName()`. Technika używana w tej bibliotece wymaga przekazania tylko nazwy elementu. Aby pobrać wszystkie znaczniki `<a>` metodą modelu DOM, trzeba użyć następującego kodu:

```
var linksList = document.getElementsByTagName('a');
```

W jQuery wystarczy poniższy zapis:

```
var linksList = $('a');
```

Uwaga: JQuery obsługuje wiele różnych selektorów. W tej książce wymieniono tylko najbardziej przydatne z nich, a pełną listę znajdziesz na stronie *http://docs.jquery.com/Selectors*.

Selektory klas

Model DOM nie udostępnia wbudowanej metody do wyszukiwania wszystkich elementów o określonej klasie, choć programiści języka JavaScript często potrzebują takiej możliwości. Załóżmy, że chcesz utworzyć pasek nawigacyjny z menu rozwijanym. Kiedy użytkownik przeniesie kursor nad jeden z głównych przycisków nawigacyjnych, powinno pojawić się menu rozwijane. Potrzebujesz sposobu, aby powiązać te przyciski z operacją rozwijania menu po przeniesieniu kursora w odpowiednie miejsce.

Uwaga: Ponieważ wyszukiwanie wszystkich elementów określonej klasy to bardzo częsta operacja, niektóre przeglądarki (między innymi najnowsze wersje Firefoksa i Safari) udostępniają tę funkcję. Jednak ponieważ nie wszystkie aplikacje tego typu ją obsługują, nieocenioną pomocą są biblioteki (na przykład jQuery), które współdziałają ze wszystkimi przeglądarkami.

Jedna z technik polega na dodaniu klasy, na przykład `navButton`, do każdego odnośnika w głównym pasku nawigacyjnym. Następnie można pobrać za pomocą kodu JavaScript odnośniki *tylko* tej klasy i powiązać je z operacją otwierania menu (na stronie 295 dowiesz się, jak to zrobić). To rozwiązanie może wydawać się skomplikowane, jednak ważne jest, że aby umożliwić działanie paska nawigacyjnego, trzeba pobrać odnośniki określonej klasy.

Na szczęście jQuery udostępnia wygodną metodę wyszukiwania wszystkich elementów danej klasy. Wystarczy użyć selektora klasy CSS:

```
$('.submenu')
```

Zauważ, że jest to standardowy selektor klasy języka CSS — wystarczy podać kropkę i nazwę klasy. Po pobraniu znaczników można manipulować nimi za pomocą jQuery. Na przykład aby ukryć wszystkie tagi klasy `.submenu`, wystarczy użyć poniższego kodu:

```
$('.submenu').hide();
```

Więcej o funkcji `hide()` jQuery dowiesz się na stronie 241, a ten wiersz ilustruje działanie omawianego mechanizmu.

Selektory zaawansowane

JQuery umożliwia używanie także bardziej skomplikowanych selektorów CSS do precyzyjnego wskazywania potrzebnych znaczników. Na tym etapie nie musisz się ich uczyć. Kiedy przeczytasz kilka następnych rozdziałów i lepiej poznasz bibliotekę jQuery oraz metody manipulowania przy jej użyciu stronami WWW, prawdopodobnie zechcesz wrócić do tego punktu i dokładniej go przeczytać.

- **Selektory potomków** umożliwiają wskazywanie znaczników zagnieżdżonych wewnątrz innych tagów (zobacz punkt „Pobieranie pobliskich węzłów" na stronie 162). Załóżmy, że utworzyłeś nieuporządkowaną listę odnośników i dodałeś do znacznika `<ul>` identyfikator `navBar`: `<ul id="navBar">`. Wyrażenie `$('a')` pozwala znaleźć za pomocą biblioteki jQuery wszystkie znaczniki `<a>` na stronie. Jednak jeśli chcesz pobrać tylko odsyłacze ze wspomnianej listy nieuporządkowanej, powinieneś użyć selektora potomków:

WIEDZA W PIGUŁCE

Style CSS

Style CSS są niezwykle istotne przy poznawaniu języka JavaScript. Aby w pełni skorzystać z tej książki, powinieneś mieć przynajmniej podstawową wiedzę z dziedziny projektowania stron WWW, znać style CSS i umieć ich używać. Język CSS to najważniejsze narzędzie projektantów stron WWW przeznaczone do tworzenia atrakcyjnych witryn, dlatego jeśli nie znasz tej technologii, powinieneś się jej nauczyć. Style CSS pomogą Ci używać biblioteki jQuery. Przekonasz się też, że przy użyciu języka JavaScript i stylów CSS można łatwo dodawać interaktywne efekty wizualne do stron WWW.

Jeśli chcesz szybko poznać język CSS, pomoże Ci w tym wiele przydatnych materiałów:

Dobrym ogólnym wprowadzeniem do języka CSS są samouczki w witrynie HTML Dog (*www.htmldog.com/guides/*). Znajdziesz tam lekcje dla początkujących, średnio zaawansowanych i doświadczonych projektantów.

Możesz też zaopatrzyć się w książkę *CSS. Nieoficjalny podręcznik*, która zawiera kompletne omówienie języka CSS (a także — podobnie jak niniejsza pozycja — liczne praktyczne przykłady).

W czasie korzystania z biblioteki jQuery bardzo ważne jest umiejętne używanie **selektorów**, czyli instrukcji informujących przeglądarkę, do których znaczników ma zastosować dany styl. Źródła podane w tej ramce pomogą Ci opanować to zagadnienie. Jeśli chcesz szybko przypomnieć sobie działanie różnych selektorów, możesz odwiedzić poniższe strony:

- *http://css.maxdesign.com.au/selectutorial/*,
- *http://www.456bereastreet.com/archive/200601/css_3_selectors_explained/*.

```
$('#navBar a')
```

Ta składnia jest oparta na języku CSS. Należy podać selektor, odstęp, a następnie kolejny selektor. Ostatni selektor (tu jest to `a`) to element docelowy, natomiast selektory umieszczone po jego lewej stronie reprezentują znaczniki obejmujące dany element.

- **Selektory dziecka** wskazują znacznik, który jest dzieckiem (czyli bezpośrednim potomkiem) innego tagu. W kodzie HTML przedstawionym na rysunku 5.2 znaczniki `<h1>` i `<p>` to dzieci tagu `<body>`. Jego dzieckiem nie jest już znacznik `<strong>`, ponieważ znajduje się w tagu `<p>`. Aby utworzyć selektor dziecka, należy najpierw podać rodzica, następnie znak `>`, a na końcu dziecko. Aby pobrać znaczniki `<p>`, które są dziećmi tagu `<body>`, można użyć następującego kodu:

  ```
  $('body > p')
  ```

- **Selektory przyległego brata** pozwalają pobrać znacznik, który pojawia się bezpośrednio po innym tagu. Załóżmy, że na stronie znajduje się niewidoczny panel, pojawiający się po kliknięciu zakładki. W kodzie HTML zakładkę może reprezentować znacznik nagłówka (na przykład `<h2>`), natomiast ukryty panel to znacznik `<div>` umieszczony bezpośrednio po nagłówku. Aby wyświetlić ten znacznik (panel), trzeba uzyskać do niego dostęp. Można to łatwo zrobić za pomocą jQuery i selektora przyległego brata:

  ```
  $('h2 + div')
  ```

  Aby zastosować tę technikę, należy wstawić znak plus między oba selektory (mogą to być selektory dowolnego typu — identyfikatorów, klas lub elementów). Biblioteka pobierze element podany po prawej stronie, jednak tylko wtedy, jeśli pojawia się bezpośrednio po selektorze określonym po lewej stronie.

- **Selektory atrybutów** umożliwiają wybór elementów o określonym atrybucie. Można nawet sprawdzić, czy atrybut ten ma konkretną wartość. Możesz na przykład znaleźć znaczniki `<img>` z atrybutem `alt`, a także o określonej wartości tego atrybutu, jak również pobrać wszystkie odnośniki prowadzące poza witrynę i sprawić, aby otwierały strony w nowym oknie.

  Aby utworzyć omawiany selektor, należy po nazwie elementu podać szukany atrybut. Na przykład aby znaleźć znaczniki `<img>` z atrybutem `alt`, możesz użyć następującego kodu:

  ```
  $('img[alt]')
  ```

  Dostępnych jest wiele rodzajów selektorów atrybutów:

  - Wersja `[atrybut]` pobiera elementy z danym atrybutem określonym w kodzie HTML. Na przykład wyrażenie `$(a[href])` znajduje wszystkie znaczniki `<a>` z atrybutem `href`. Pozwala to pominąć kotwice z nazwami (na przykład `<a name="placeOnPage"></a>`), które pełnią funkcję odnośników wewnątrz strony.
  - Wersja `[atrybut=wartość]` pobiera elementy, w których dany atrybut ma określoną wartość. Aby znaleźć w formularzu wszystkie pola tekstowe, można użyć następującego kodu:

    ```
    $('input[type=text]')
    ```

    Ponieważ większość elementów formularza ma ten sam znacznik, `<input>`, jedyny sposób na wskazanie konkretnych tagów polega na użyciu atrybutu `type`. Pobieranie elementów formularza to tak częsta operacja, że jQuery udostępnia specjalne selektory do wykonywania tej operacji. Poznasz je na stronie 305.

  - Wersja `[atrybut^=wartość]` pasuje do elementów o atrybucie, który rozpoczyna się od określonej wartości. Jeśli chcesz znaleźć odnośniki prowadzące poza witrynę, możesz użyć poniższego wyrażenia:

    ```
    $('a[href^=http://]')
    ```

    Zauważ, że sprawdzany jest tylko początek atrybutu, a nie cała jego wartość. Dlatego wyrażenie `href^=http://` pasuje do adresów *http://www.yahoo.com*, *http://www.google.com* i tak dalej. Przy użyciu tego selektora możesz też pobrać odnośniki typu `mailto:`, na przykład:

    ```
    $('a[href^=mailto:]')
    ```

  - Wersja `[atrybut$=wartość]` pasuje do elementów, których atrybut kończy się określonym tekstem. Jest to doskonała technika wyszukiwania plików o danym rozszerzeniu. Przy użyciu tego selektora możesz na przykład znaleźć odnośniki prowadzące do plików PDF (aby dodać do nich specjalne ikony za pomocą kodu JavaScript lub dynamicznie wygenerować odnośnik do witryny *Adobe.com*, skąd użytkownicy mogą pobrać program Acrobat Reader). Kod pobierający adresy plików PDF wygląda następująco:

    ```
    $('a[href$=.pdf]')
    ```

  - Wersja `[atrybut*=wartość]` pasuje do elementów, których atrybut zawiera określoną wartość w dowolnym miejscu. Można na przykład pobrać wszystkie odnośniki prowadzące do danej domeny. Poniższy kod znajduje odsyłacze do domeny *misingmanuals.com*:

```
$('a[href*=missingmanuals.com]')
```

Ten selektor zapewnia dużą elastyczność, ponieważ umożliwia znalezienie odnośników kierujących użytkownika pod adres *http://www.missingmanuals.com*, ale też prowadzących do stron *http://missingmanuals.com* i *http://www.missingmanuals.com/library.html*.

Uwaga: Biblioteka jQuery udostępnia zestaw selektorów przydatnych przy korzystaniu z formularzy. Selektory te umożliwiają wyszukiwanie między innymi pól tekstowych, pól z hasłami i zaznaczonych przycisków opcji. Więcej informacji o tym zestawie znajdziesz na stronie 305.

Filtry biblioteki jQuery

JQuery umożliwia filtrowanie pobranych elementów na podstawie określonych cech. Filtr `:even` pozwala pobrać wszystkie parzyste elementy kolekcji (użyłeś go w przykładzie na stronie 42 do wyróżnienia co drugiego wiersza tabeli). Ponadto można wyszukać elementy zawierające określony odnośnik lub tekst, znaczniki ukryte, a nawet tagi, które **nie** pasują do podanego selektora. Aby użyć filtra, należy dodać dwukropek przed jego nazwą, a po podstawowym selektorze. Na przykład aby pobrać wszystkie parzyste wiersze tabeli, należy użyć poniższego selektora jQuery:

```
$('tr:even')
```

Ten kod pobiera każdy parzysty znacznik `<tr>`. Aby zawęzić zakres pobieranych elementów, można zażądać pobrania co drugiego wiersza tabel klasy `striped`:

```
$('.striped tr:even')
```

Poniżej znajdziesz opis działania instrukcji `:even` i innych filtrów:

- Filtry `:even` i `:odd` pobierają *co drugi* element kolekcji. Te filtry działają dość nieintuicyjnie. Pamiętaj, że jQuery wyszukuje wszystkie elementy, które pasują do podanego selektora. Pobierane dane mają formę tablicy (zobacz stronę 68), a każdy element ma indeks (wartości indeksów rozpoczynają się od `0`; strona 71). Dlatego ponieważ filtr `:even` pobiera elementy o indeksach parzystych (`0`, `2`, `4` i tak dalej), zwraca pierwszą, trzecią, piątą i następne wartości, czyli wszystkie elementy nieparzyste! Filtr `:odd` działa w podobny sposób, jednak pobiera elementy o indeksach nieparzystych (`1`, `3`, `5` i tak dalej).
- Można użyć polecenia `:not()`, aby pobrać elementy, które **nie** pasują do podanego selektora. Załóżmy, że chcesz pobrać wszystkie znaczniki `<a>` oprócz tagów klasy `navButton`. Można to zrobić za pomocą poniższego kodu:

  ```
  $('a:not(.navButton)');
  ```

  Do funkcji `:not()` należy przekazać nazwę pomijanego selektora. Tu `.navButton` to selektor klasy, dlatego podany kod oznacza: „Elementy, które nie mają klasy `.navButton`". Funkcji `:not()` można używać razem z innymi filtrami biblioteki jQuery i z większością selektorów. Aby znaleźć wszystkie odnośniki, które nie rozpoczynają się od członu „http://", należy użyć następującego kodu:

  ```
  $('a:not([href^=http://])')
  ```

- Filtr :has() znajduje elementy zawierające podany selektor. Załóżmy, że chcesz pobrać wszystkie znaczniki <li>, które zawierają element <a>. Można to zrobić w następujący sposób:

  ```
  $('li:has(a)')
  ```

  To rozwiązanie różni się od selektora potomka, ponieważ nie powoduje pobrania znacznika <a>, ale elementu <li> zawierającego odnośnik.

- Filtr :contains() wyszukuje elementy zawierające określony tekst. Aby znaleźć wszystkie odnośniki z napisem „Kliknij mnie!", możesz utworzyć następujący obiekt jQuery:

  ```
  $('a:contains(Kliknij mnie!)')
  ```

- Filtr :hidden wyszukuje elementy ukryte. Są to znaczniki mające właściwość CSS display o wartości none (co powoduje, że nie są widoczne na stronie), ukryte za pomocą funkcji hide() biblioteki jQuery (opis tej funkcji znajdziesz na stronie 241) i ukryte pola formularza. Ten selektor nie uwzględnia elementów mających właściwość CSS visibility o wartości invisible. Załóżmy, że ukryłeś kilka znaczników . Za pomocą biblioteki jQuery możesz je znaleźć i wyświetlić:

  ```
  $('div:hidden').show();
  ```

  Ten wiersz kodu nie wpływa na znaczniki widoczne na stronie. Funkcję show() biblioteki jQuery szczegółowo opisano na stronie 241.

- Filtr :visible to przeciwieństwo filtra :hidden — pozwala znaleźć elementy widoczne na stronie.

Kolekcje elementów pobranych za pomocą jQuery

Elementy pobrane za pomocą obiektu jQuery (takiego jak $('#navBar a')) nie mają postaci tradycyjnych węzłów modelu DOM, zwracanych przez polecenia getElementById() i getElementsByTagName(). W zamian dostępne są kolekcje elementów specyficzne dla jQuery. Te elementy nie obsługują tradycyjnych metod modelu DOM. Nie można na przykład użyć właściwości innerHTML (strona 165):

```
$('#banner').innerHTML = 'Nowy tekst'; // Ten kod nie zadziała.
```

Jeśli poznałeś metody modelu DOM, czytając inną książkę, odkryjesz, że żadna z nich nie działa dla standardowych obiektów jQuery. Na pozór jest to poważna wada, jednak prawie wszystkie właściwości i metody modelu DOM mają swe odpowiedniki w bibliotece jQuery, dlatego przy jej użyciu można wykonać wszystkie standardowe operacje, a w dodatku zwykle dużo szybciej i w mniejszej liczbie wierszy.

Występują jednak dwie istotne różnice między działaniem modelu DOM i kolekcji elementów jQuery. Bibliotekę jQuery zbudowano w celu ułatwienia i przyspieszenia programowania w języku JavaScript. Jednym z celów było umożliwienie wykonywania wielu operacji w jak najmniejszej liczbie wierszy kodu. Aby to osiągnąć, zastosowano dwa niestandardowe rozwiązania.

Pętle automatyczne

W modelu DOM po pobraniu kolekcji węzłów trzeba zwykle utworzyć pętlę (strona 98), przejść w niej po wszystkich znalezionych elementach i wykonać na nich określone operacje. Jeśli chcesz pobrać wszystkie rysunki, a następnie je ukryć (na przykład przy tworzeniu pokazu slajdów w języku JavaScript), musisz najpierw znaleźć obrazki i utworzyć pętlę, która przejdzie po ich liście.

Ponieważ przechodzenie w pętli po kolekcjach jest standardową operacją, funkcje jQuery robią to domyślnie. Oznacza to, że jeśli wywołasz taką funkcję dla kolekcji elementów, nie musisz tworzyć pętli, ponieważ biblioteka zrobi to za Ciebie.

Aby za pomocą jQuery pobrać wszystkie rysunki ze znacznika `<div>` o identyfikatorze `slideshow`, a następnie je ukryć, możesz użyć następującego kodu:

```
$('#slideshow img').hide();
```

Lista elementów pobrana za pomocą instrukcji `$('#slideshow img')` może zawierać 50 rysunków. Funkcja `hide()` automatycznie przejdzie w pętli po elementach kolekcji i ukryje każdy obrazek. To rozwiązanie jest tak wygodne (wyobraź sobie liczbę pętli `for`, których nie będziesz musiał pisać), że zaskakujące jest, iż technika ta nie jest częścią modelu DOM.

Łańcuchy funkcji

Czasem program ma wykonać kilka operacji na kolekcji elementów. Załóżmy, że chcesz za pomocą kodu JavaScript ustawić szerokość i wysokość znacznika `<div>` o identyfikatorze `popUp`. Zwykle wymaga to napisania przynajmniej dwóch wierszy kodu, jednak jQuery umożliwia użycie tylko jednej instrukcji:

```
$('#popUp').width(300).height(300);
```

JQuery obsługuje *łańcuchy funkcji*. Jest to wyjątkowy mechanizm, który umożliwia podawanie funkcji jedna po drugiej. Funkcje są połączone ze sobą znakiem kropki i wszystkie działają na tej samej kolekcji elementów. Dlatego powyższy kod zmienia szerokość elementu o identyfikatorze `popUp` **oraz** jego wysokość. Łańcuchy funkcji jQuery umożliwiają przeprowadzanie wielu operacji za pomocą zwięzłego kodu. Załóżmy, że chcesz nie tylko podać szerokość i wysokość elementu `<div>`, ale też dodać do niego tekst i stopniowo wyświetlać na stronie (przyjmujemy, że nie jest jeszcze widoczny). Możesz to zrobić w bardzo zwięzły sposób:

```
$('#popUp').width(300).height(300).text('Hej!').fadeIn(1000);
```

Ten kod uruchamia cztery funkcje (`width()`, `height()`, `text()` i `fadeIn()`) dla znacznika o identyfikatorze `popUp`.

Wskazówka: Długi łańcuch funkcji jQuery może być mało czytelny, dlatego niektórzy programiści dzielą takie struktury na kilka wierszy:

```
$('#popUp').width(300)
   .height(300)
   .text('Hej!')
   .fadeIn(1000);
```

Jeśli umieścisz średnik w *ostatnim wierszu* łańcucha, interpreter potraktuje wszystkie polecenia jak jedną instrukcję.

Możliwość tworzenia łańcuchów funkcji jest niestandardowa i specyficzna dla biblioteki jQuery. Oznacza to, że w łańcuchu nie można umieszczać funkcji spoza tej biblioteki (utworzonych samodzielnie lub wbudowanych poleceń języka JavaScript).

Dodawanie treści do stron

JQuery udostępnia wiele funkcji do manipulowania elementami i zawartością stron — od prostego zastępowania kodu HTML, przez precyzyjne rozmieszczanie nowych elementów HTML w stosunku do wybranego znacznika, po całkowite usuwanie tagów i treści ze stron.

Uwaga: Przykładowy plik *content_functions.html* z katalogu *R05* umożliwia przetestowanie wybranych funkcji jQuery. Otwórz ten plik w przeglądarce, wpisz tekst w odpowiednim polu, a następnie kliknij jedno z pól *Uruchom*, aby zobaczyć, jak działa dana funkcja.

Na potrzeby opisu funkcji załóżmy, że strona zawiera następujący kod HTML:

```
<div id="container">
  <div id="errors">
    <h2>Błędy:</h2>
  </div>
</div>
```

- Funkcja `.html()` działa podobnie jak właściwość `innerHTML` modelu DOM. Pozwala pobrać kod HTML danego elementu, a także zastąpić go innym fragmentem. Funkcji tej należy używać dla elementów pobranych za pomocą jQuery.

  Aby pobrać kod HTML danego elementu, dodaj po jego nazwie wywołanie `.html()`. Na przykładowej stronie można wywołać następujące polecenie:

  ```
  alert($('#errors').html());
  ```

  Ten kod wyświetla okno dialogowe z tekstem „`<h2>Błędy:</h2>`". Przy używaniu funkcji `html()` w taki sposób można skopiować kod HTML z danego elementu i wkleić go w innym znaczniku.

- Aby zastąpić bieżącą zawartość określonego elementu, należy przekazać do metody `.html()` argument w postaci łańcucha znaków:

  ```
  $('#errors').html('<p>Formularz zawiera cztery błędy</p>');
  ```

  Ten wiersz zastępuje cały kod HTML elementu o identyfikatorze `errors`. Po tym wywołaniu przykładowy fragment kodu HTML będzie wyglądał następująco:

  ```
  <div id="containers">
    <div id="errors">
      <p>Formularz zawiera cztery błędy</p>
    </div>
  </div>
  ```

  Zauważ, że użyta funkcja zastąpiła znacznik `<h2>` z pierwszej wersji kodu. Inne funkcje pozwalają tego uniknąć.

Uwaga: JQuery udostępnia też funkcję o nazwie `text()`, która działa podobnie jak `html()`. Różnica polega na tym, że znaczniki HTML przekazywane do funkcji `text()` są kodowane, na przykład `<p>` przyjmuje formę `&lt;p&gt;`. Tej funkcji należy używać do wyświetlania nawiasów ostrych i nazw znaczników *w tekście* strony. Przy jej użyciu możesz na przykład przedstawić internautom fragment kodu HTML.

- Funkcja `append()` dodaje kod HTML jako ostatnie dziecko pobranego znacznika. Załóżmy, że pobrałeś znacznik `<div>`, jednak zamiast zastępować jego zawartość, chcesz dodać kod HTML przed zamykającym znacznikiem `</div>`. Funkcja `.append()` to doskonałe narzędzie do dodawania elementów na koniec list wypunktowanych (`<ul>`) i numerowanych (`<ol>`). Na stronie z przykładowym fragmentem kodu HTML można uruchomić poniższe wywołanie:

```
$('#errors').append('<p>Formularz zawiera cztery błędy</p>');
```

  Po wykonaniu tej funkcji kod HTML będzie wyglądał następująco:

```
<div id="containers">
  <div id="errors">
    <h2>Błędy:</h2>
    <p>Formularz zawiera cztery błędy</p>
  </div>
</div>
```

  Zauważ, że w znaczniku `<div>` pozostał pierwotny kod HTML, a wywołanie dodało do niego nowy fragment strony.

- Funkcja `prepend()` działa podobnie jak `append()`, ale dodaje kod HTML bezpośrednio po otwierającym znaczniku pobranego elementu. Na stronie z przykładowym fragmentem kodu HTML można uruchomić poniższe wywołanie:

```
$('#errors').prepend('<p>Formularz zawiera cztery błędy</p>');
```

  Po wywołaniu funkcji `prepend()` przykładowy kod HTML będzie wyglądał następująco:

```
<div id="containers">
  <div id="errors">
    <p>Formularz zawiera cztery błędy</p>
    <h2>Błędy:</h2>
  </div>
</div>
```

  Teraz nowa treść pojawia się bezpośrednio po otwierającym znaczniku `<div>`.

- Jeśli chcesz dodać kod HTML *poza* pobranym elementem (bezpośrednio przed znacznikiem otwierającym lub po znaczniku zamykającym), możesz użyć funkcji `before()` i `after()`. Powszechną praktyką jest sprawdzanie, czy pola tekstowe przesyłanego formularza nie są puste. Przed przesłaniem formularza kod HTML pola wygląda następująco:

```
<input type="text" name="userName" id="userName">
```

  Przyjmijmy, że kiedy użytkownik chce przesłać formularz, to pole jest puste. Można napisać program, który sprawdzi zawartość tego pola, a następnie doda komunikat o błędzie. Aby wyświetlić informację pod polem (w tym miejscu nie jest istotne, czy sprawdzanie zawartości pola przebiega poprawnie — technikę tę poznasz na stronie 323), możesz użyć funkcji `.after()`:

```
$('userName').after('<span class="error">Podaj nazwę
użytkownika</span>');
```

Ten wiersz kodu powoduje wyświetlenie na stronie komunikatu o błędzie. Kod HTML będzie wyglądał następująco:

```
<input type="text" name="userName" id="userName">
<span class="error">Podaj nazwę użytkownika</span>
```

Funkcja `.before()` umieszcza nową treść przed pobranym elementem.

Zastępowanie i usuwanie pobranych elementów

Czasem programista chce całkowicie zastąpić lub usunąć pobrany element. Załóżmy, że utworzyłeś za pomocą języka JavaScript wyskakujące okno dialogowe (nie standardowe okienko wyświetlane przez polecenie `alert()`, ale profesjonalnie wyglądającą kontrolkę, opartą na elemencie `<div>` wyświetlanym w górnej części strony). Kiedy użytkownik kliknie przycisk *Zamknij* tego okna, należy usunąć je ze strony. Możesz użyć do tego funkcji `remove()` biblioteki jQuery. Jeśli okno ma identyfikator `popup`, można je usunąć za pomocą poniższego kodu:

```
$('#popup').remove();
```

Funkcja `.remove()` działa też dla większej liczby elementów. Jeśli chcesz usunąć wszystkie znaczniki `<span>` klasy `error`, możesz to zrobić w następujący sposób:

```
$('span.error').remove();
```

PRZYDATNE NARZĘDZIA

Podgląd zmian za pomocą dodatku View Source Chart

Przy manipulowaniu elementami modelu DOM w kodzie JavaScript występuje pewien problem. Do czasu przygotowania całego skryptu JavaScript trudno jest określić, jak będzie wyglądał kod HTML po dodaniu, zmodyfikowaniu, usunięciu lub nowym rozmieszczeniu jego fragmentów. Polecenie *Pokaż kod źródłowy*, dostępne w każdej przeglądarce, wyświetla plik w formie pobranej z serwera. Oznacza to, że zobaczysz kod HTML *sprzed* jego zmodyfikowania przez kod JavaScript. Utrudnia to ustalenie, czy rozwijany program JavaScript naprawdę generuje pożądany kod HTML. Byłoby to dużo łatwiejsze, gdybyś mógł zobaczyć, jak kod HTML strony będzie wyglądał po dodaniu do formularza 10 komunikatów o błędach lub po utworzeniu długiego okna dialogowego z tekstem i polami formularza.

Na szczęście istnieje kilka dodatków dla przeglądarki Firefox, które pomagają rozwiązać ten problem. Jedno z nich, *View Source Chart* (*http://jennifermadden.com/scripts/ViewRenderedSource.html*), pozwala zobaczyć aktualny stan modelu DOM. Oznacza to, że jeśli otworzysz okno tego narzędzia po dodaniu lub zmodyfikowaniu kodu HTML przez skrypt JavaScript, zobaczysz nowy, zmieniony dokument HTML.

Aby użyć tego dodatku, otwórz Firefoksa, przejdź pod podany wcześniej adres i zainstaluj dodatek. Kiedy następnie zechcesz wyświetlić aktualny kod HTML strony, wybierz opcję *View Source Chart* z menu *Widok* przeglądarki. Jeśli później program JavaScript spowoduje następne zmiany w modelu DOM (na przykład w wyniku kliknięcia rysunku lub próby przesłania formularza), to aby wyświetlić nowo wygenerowany kod HTML, musisz zamknąć okno *View Source Chart* i ponownie je otworzyć.

Inne rozszerzenie, Web Developer Toolbar (*http://chrispederick.com/work/web-developer/*), udostępnia podobne narzędzie. Przejdź w przeglądarce Firefox pod podany adres i zainstaluj ten dodatek. Po ponownym uruchomieniu przeglądarki zobaczysz wiele nowych opcji w pasku narzędzi (rozszerzenie to udostępnia **wiele** przydatnych narzędzi dla programistów witryn). Jeśli wybierzesz polecenie *View Source/View Generated Source*, zobaczysz model DOM zmodyfikowany przez kod JavaScript. Jednak kod HTML wyświetlany przez to narzędzie nie jest tak dobrze sformatowany jak w dodatku View Source Chart, dlatego nie jest równie czytelny.

Można też całkowicie zastąpić pobrany element nową treścią. Załóżmy, że strona zawiera zdjęcia sprzedawanych produktów. Kiedy użytkownik kliknie fotografię, skrypt dodaje towar do koszyka. Można wtedy zastąpić znacznik `<img>` tekstem, na przykład „Dodano do koszyka". W następnym rozdziale dowiesz się, jak sprawić, aby program reagował na zdarzenia związane ze specyficznymi elementami (na przykład na kliknięcie rysunku). Na razie załóżmy, że chcesz zastąpić tekstem znacznik `<img>` o identyfikatorze `product101`. Za pomocą jQuery można to zrobić w następujący sposób:

```
$('#product101').replace('<p>Dodano do koszyka</p>');
```

Ten kod usuwa ze strony znacznik `<img>` i zastępuje go elementem `<p>`.

Uwaga: JQuery udostępnia też funkcję `clone()`, która kopiuje pobrany element. Tej funkcji użyjesz w przykładzie na stronie 198.

Ustawianie i wczytywanie atrybutów

Dodawanie, usuwanie i modyfikowanie elementów to nie jedyne operacje obsługiwane przez jQuery (i nie jedyne, które programiści chcą przeprowadzać na pobranych elementach). Często przydatna jest zmiana atrybutu elementu — dodanie klasy do znacznika lub zmiana jego właściwości CSS. Można też sprawdzić wartość atrybutu i na przykład ustalić, pod jaki adres URL kieruje użytkowników dany odnośnik.

Klasy

Technologia CSS ma bardzo duże możliwości i pozwala na zaawansowane wizualnie formatowanie kodu HTML. Jedna reguła CSS może dodawać kolorowe tło do strony, a inna — całkowicie ukrywać element. Używając języka JavaScript do usuwania, dodawania i modyfikowania klas, można uzyskać naprawdę złożone efekty wizualne. Ponieważ przeglądarki przetwarzają instrukcje CSS bardzo szybko i wydajnie, samo dodanie klasy do znacznika pozwala zupełnie zmienić jego wygląd, a nawet ukryć go.

W przykładzie na stronie 42 napisałeś program JavaScript, który zmieniał kolor tła co drugiego wiersza tabeli. Na zapleczu skrypt ten przypisywał określoną klasę do odpowiednich wierszy. Za kolor tła odpowiadał styl CSS.

JQuery udostępnia kilka funkcji do manipulowania atrybutem `class` znaczników:

- Funkcja `addClass()` dodaje do elementu określoną klasę. Przed wywołaniem należy podać pobrany element, a do funkcji trzeba przekazać łańcuch znaków z nazwą dodawanej klasy. Aby dodać klasę `externalLink` do wszystkich odnośników prowadzących poza witrynę, można użyć poniższego kodu:

  ```
  $('a[href^=http://]').addClass('externalLink');
  ```

  Ten kod pobiera elementy HTML podobne do poniższego:

  ```
  <a href="http://www.oreilly.com/">
  ```

I przekształca je na następującą postać:

```
<a href="http://www.oreilly.com/" class="externalLink">
```

Aby ta funkcja była przydatna, trzeba wcześniej przygotować w arkuszu styl klasy CSS. Kiedy następnie kod JavaScript doda nazwę klasy, przeglądarka zastosuje do odpowiednich elementów właściwości zdefiniowanego wcześniej stylu CSS.

Uwaga: Funkcje `addClass()` i `removeClass()` przyjmują jako argument samą nazwę klasy. Należy pominąć kropkę używaną przy tworzeniu selektorów klas. Zapis `addClass('externalLink')` jest poprawny, natomiast forma `addClass('.externalLink')` jest nieprawidłowa.

Jeśli dany znacznik ma już atrybut `class`, funkcja `addClass()` nie usuwa klas wcześniej przypisanych do elementu, a jedynie dodaje nową klasę.

Uwaga: Przypisanie wielu klas do jednego znacznika jest w pełni dozwolone i często bardzo przydatne. Więcej informacji o tej technice znajdziesz na stronie *http://webdesign.about.com/od/css/qt/tipcssmulticlas.htm*.

- Funkcja `removeClass()` to przeciwieństwo polecenia `addClass()` — usuwa określoną klasę z pobranego elementu. Aby usunąć klasę `highlight` z elementu `<div>` o identyfikatorze `alertBox`, możesz użyć następującego kodu:

  ```
  $('#alertBox').removeClass('highlight');
  ```

- Możesz też chcieć *przełączyć* klasę (czyli dodać ją, jeśli element jej nie ma, lub usunąć, jeżeli program przypisał ją wcześniej do znacznika). Przełączanie to bardzo popularna metoda wyświetlania elementów w trybie włączonym i wyłączonym. Kiedy klikniesz przycisk opcji, zaznaczysz go (włączanie). Gdy klikniesz go ponownie, znak zaznaczenia zniknie (wyłączanie).

  Załóżmy, że na stronie znajduje się przycisk, którego kliknięcie powoduje zmianę klasy znacznika `<body>`. Można w ten sposób całkowicie zmienić styl strony przy użyciu drugiego zestawu stylów i selektorów potomków. Kiedy użytkownik ponownie kliknie przycisk, należy usunąć klasę z tagu `<body>` i przywrócić pierwotny wygląd strony. Przyjmijmy, że przycisk zmieniający styl strony ma identyfikator `changeStyle` i przełącza klasę `altStyle`:

  ```
  $('#changeStyle').click(function() {
    $('body').toggleClass('altStyle');
  });
  ```

  Na razie nie musisz wiedzieć, jak działa pierwszy i trzeci wiersz kodu. Są one związane ze zdarzeniami (rozdział 6.), które umożliwiają reagowanie skryptu na zjawiska zachodzące na stronie (na przykład na kliknięcie przycisku). Wiersz wyróżniony pogrubieniem pokazuje, jak użyć funkcji `toggleClass()`. Przy każdym kliknięciu przycisku dodaje ona lub usuwa klasę `altStyle`.

Wczytywanie i modyfikowanie właściwości CSS

Funkcja `css()` umożliwia bezpośrednie zmienianie właściwości CSS elementów. Dlatego zamiast dodawać do znacznika klasę, można natychmiast zmodyfikować jego obramowanie, kolor tła, szerokość lub pozycję. Funkcji `css()` można używać na trzy sposoby: do sprawdzania bieżącej wartości właściwości CSS elementu, do ustawiania wartości pojedynczych właściwości CSS znaczników lub do jednoczesnej zmiany wielu właściwości.

Aby ustalić aktualną wartość właściwości CSS, należy przekazać jej nazwę do funkcji `css()`. Poniższy kod pobiera kolor tła znacznika `<div>` o identyfikatorze `main`:

```
var bgColor = $('#main').css('background-color');
```

Uruchomienie tego kodu spowoduje zapisanie w zmiennej `bgColor` łańcucha znaków określającego kolor tła elementu. JQuery zwróci albo wartość `'transparent'` (jeśli znacznik nie ma ustawionego koloru tła), albo kolor zapisany jako wartości RGB, na przykład `'rgb(255, 255, 0)'`.

Uwaga: JQuery nie zawsze zwraca wartości CSS w oczekiwany sposób. Biblioteka nie obsługuje właściwości skróconych, na przykład `font`, `margin`, `padding` i `border`. W zamian trzeba użyć pełnych form — `font-face`, `margin-top`, `padding-bottom` lub `border-bottom-width`. Ponadto jQuery przekształca wszystkie jednostki na piksele, dlatego nawet jeśli użyjesz stylów CSS do ustawienia rozmiaru czcionki znacznika `<body>` na `150%`, jQuery zwróci wartość właściwości `font-size` w pikselach.

Funkcja `css()` umożliwia też ustawianie właściwości CSS elementów. Aby użyć jej w ten sposób, należy podać dwa argumenty: nazwę właściwości CSS i jej wartość. Poniższy wiersz zmienia rozmiar czcionki znacznika `<body>` na `200%`:

```
$('body').css('font-size', '200%');
```

Drugi argument może być łańcuchem znaków (na przykład `'200%'`) lub liczbą. Wartości liczbowe jQuery przekształca na piksele. Jeśli chcesz zmienić marginesy wewnętrzne wszystkich znaczników klasy `.pullquote` na 100 pikseli, możesz użyć poniższego kodu:

```
$('.pullquote').css('padding',100);
```

Biblioteka jQuery przypisze do właściwości `padding` wartość 100 pikseli.

Uwaga: Przy ustawianiu właściwości CSS za pomocą funkcji `.css()` biblioteki jQuery można używać zapisu skróconego. Na przykład aby dodać czarne, jednopikselowe obramowanie do wszystkich akapitów klasy `highlight`, wystarczy wywołać poniższą instrukcję:

```
$('p.highlight').css('border', '1px solid black');
```

Często zakres zmian właściwości CSS zależy od ich aktualnej wartości. Na stronie można umieścić przycisk *Powiększ tekst*, którego kliknięcie powoduje podwojenie rozmiaru czcionki. Aby zmienić tę wielkość, trzeba najpierw sprawdzić bieżący rozmiar czcionki, a następnie go podwoić. Jest to nieco bardziej skomplikowane, niż się na pozór wydaje. Oto potrzebny kod i jego opis:

```
var baseFont = $('body').css('font-size');
baseFont = parseInt(baseFont,10);
$('body').css('font-size',baseFont * 2);
```

Pierwszy wiersz pobiera wartość właściwości `font-size` znacznika `<body>`. Wartość ta jest określona w pikselach i ma postać łańcucha znaków, na przykład `'16px'`. Ponieważ skrypt ma podwoić rozmiar czcionki, trzeba przekształcić łańcuch na liczbę, usuwając człon „px". Odpowiada za to drugi wiersz, w którym użyto omówionej na stronie 140 metody `parseInt()` języka JavaScript. Funkcja ta usuwa tekst następujący po liczbie, dlatego po wykonaniu drugiego wiersza zmienna `baseFont` zawiera liczbę, na przykład 16. Trzeci wiersz zmienia wartość właściwości `font-size` na wartość zmiennej `baseFont`, pomnożoną przez 2.

Uwaga: Ten kod zmienia rozmiar czcionki na stronie tylko wtedy, jeśli wielkość znaków pozostałych znaczników — akapitów, nagłówków i tak dalej — jest określona za pomocą wartości względnych, na przykład jednostek em lub procentów. Modyfikacja rozmiaru czcionki znacznika `<body>` nie wpływa na tagi z wielkością znaków ustawioną za pomocą jednostek bezwzględnych, na przykład pikseli.

Jednoczesna zmiana wielu właściwości CSS

Jeśli chcesz zmienić więcej niż jedną właściwość CSS elementu, nie musisz wielokrotnie wywoływać funkcji `.css()`. Na przykład aby dynamicznie wyróżnić znacznik `<div>` w odpowiedzi na działania użytkownika, trzeba zmienić kolor tła *oraz* obramowanie tego elementu:

```
$('#highlightedDiv').css('background-color', '#FF0000');
$('#highlightedDiv').css('border','2px solid #FE0037');
```

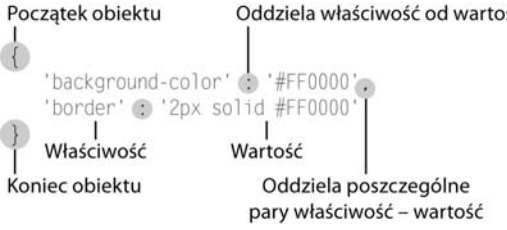
Inny sposób polega na przekazaniu do funkcji `.css()` *literału obiektowego*. Jest to lista par nazwa – wartość właściwości. Po każdej nazwie właściwości należy wpisać dwukropek (`:`), a następnie wartość. Poszczególne pary trzeba rozdzielić przecinkami, a całą listę należy umieścić w nawisach klamrowych, `{}`. Literał obiektowy ustawiający dwie wspomniane wcześniej właściwości CSS wygląda następująco:

```
{ 'background-color' : '#FF0000', 'border' : '2px solid #FE0037' }
```

Ponieważ literał obiektowy zapisany w jednym wierszu może być nieczytelny, wielu programistów woli podzielić tę strukturę. Poniższy zapis działa tak samo jak poprzedni wiersz:

```
{
  'background-color' : '#FF0000',
  'border' : '2px solid #FE0037'
}
```

Podstawową strukturę literałów obiektowych ilustruje rysunek 5.7.



Rysunek 5.7. *Literały obiektowe języka JavaScript umożliwiają tworzenie list właściwości i ich wartości. JavaScript traktuje taki literał jak pojedynczy blok informacji, podobnie jak tablicę, która jest listą wartości. Przy ustawianiu opcji na potrzeby wtyczek biblioteki jQuery często będziesz używał takich literałów*

Ostrzeżenie: Przy tworzeniu literałów obiektowych pamiętaj o rozdzielaniu par nazwa – wartość za pomocą przecinków (w przykładowym literale taki przecinek znajduje się po wartości `'#FF0000'`). *Nie* należy jednak umieszczać przecinka po ostatniej parze właściwość – wartość. Jeśli wstawisz przecinek w tym miejscu, niektóre przeglądarki (w tym Internet Explorer) zgłoszą błąd.

Aby użyć literału obiektowego w funkcji `css()`, wystarczy przekazać go jako argument:

```
$('#highlightedDiv').css({
  'background-color' : '#FF0000',
  'border' : '2px solid #FE0037'
});
```

Przyjrzyj się uważnie temu fragmentowi. Wygląda nieco inaczej od wcześniejszych wywołań, a w dalszych rozdziałach często spotkasz się z takim zapisem. Przede wszystkim warto zauważyć, że jest to jedna instrukcja języka JavaScript (jeden wiersz kodu). Informuje o tym średnik, który pojawia się dopiero w ostatnim wierszu. Instrukcja jest podzielona na cztery wiersze, co poprawia czytelność kodu.

Zauważ też, że literał obiektowy to argument (pojedynczy blok danych) przekazywany do funkcji `css()`. W kodzie `css({` otwierający nawias jest częścią funkcji, natomiast początkowy nawias klamrowy rozpoczyna kod obiektu. Trzy znaki w ostatnim wierszu to: `}`, który wyznacza koniec obiektu i argumentu przekazanego do funkcji, `)`, określający koniec funkcji `css()` (to ostatni nawias tej funkcji), i `;`, oznaczający koniec instrukcji języka JavaScript.

Jeśli zrozumienie literałów obiektowych będzie Ci sprawiać problemy, zawsze możesz zmieniać właściwości CSS pojedynczo:

```
$('#highlightedDiv').css('background-color', '#FF0000');
$('#highlightedDiv').css('border','2px solid #FE0037');
```

Wczytywanie, ustawianie i usuwanie atrybutów HTML

Ponieważ zmienianie klas i właściwości CSS za pomocą kodu JavaScript to często wykonywane zadania, jQuery udostępnia specjalne funkcje do ich obsługi. Jednak metody `addClass()` i `css()` pozwalają tylko w szybszy sposób zmienić atrybuty `class` i `style` języka HTML. JQuery udostępnia też bardziej uniwersalne funkcje do zarządzania atrybutami w kodzie HTML. Te metody to `attr()` i `removeAttr()`.

Funkcja `attr()` pozwala pobrać wartość określonego atrybutu HTML znacznika. Aby sprawdzić, na jaki plik graficzny wskazuje dany tag `<img>`, należy przekazać do wspomnianej funkcji łańcuch znaków `'src'` (oznacza on właściwość `src` znacznika):

```
var imageFile = $('#banner img').attr('src');
```

Funkcja `attr()` zwraca wartość atrybutu określoną w kodzie HTML. Powyższy wiersz zwróci wartość właściwości `src` pierwszego znacznika `<img>` w tagu o identyfikatorze `banner`, dlatego zmienna `imageFile` będzie zawierać ścieżkę podaną w kodzie HTML strony, na przykład `'images/banner.png'` lub `'http://www.thesite.com/images/banner.png'`.

Uwaga: Przy przekazywaniu nazw atrybutów do funkcji `.attr()` wielkość znaków nie jest istotna. Poprawny jest dowolny zapis: `href`, `HREF` lub `HrEf`.

Aby ustawić wartość atrybutu, należy przekazać do funkcji `attr()` także drugi argument. Jeśli chcesz wyświetlić nowy rysunek, możesz zmienić właściwość `src` znacznika `<img>`:

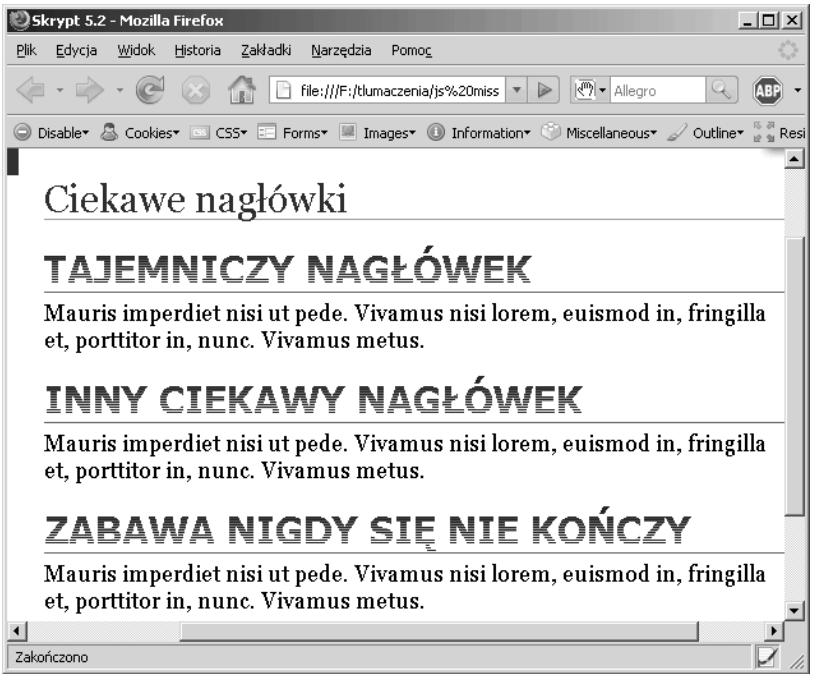
```
$('#banner img').attr('src','images/newImage.png');
```

Możesz też usunąć atrybut ze znacznika. Służy do tego funkcja `removeAttr()`. Poniższy kod usuwa właściwość `bgColor` ze znacznika `<body>`:

```
$('body').removeAttr('bgColor');
```

Ciekawe nagłówki

W tym przykładzie użyjesz biblioteki jQuery i stylów CSS do utworzenia wyjątkowego efektu związanego z nagłówkami (rysunek 5.8). Pomysł polega na przesłonięciu każdego nagłówka półprzezroczystym rysunkiem PNG. Działa on jak maska, która zakrywa fragmenty tekstu. W tym przykładzie umieścisz rysunek składający się z półprzezroczystych poziomych linii nad kilkoma nagłówkami, które będą wyglądać, jakby miały paski.



Rysunek 5.8. *Przy użyciu pomysłowego kodu CSS i JavaScript łatwo jest zwiększyć atrakcyjność wizualną nagłówków i tekstu. Aby zobaczyć więcej przykładów wykorzystania tego efektu i przeczytać o jego działaniu, odwiedź stronę* www.webdesignerwall.com/tutorials/css-gradient-text-effect *lub* http://cssglobe.com/lab/textgradient

Istotą tego efektu jest pusty znacznik `<span>` wewnątrz każdego nagłówka. Przy użyciu CSS można sformatować znacznik `<span>`, aby był widoczny nad nagłówkiem i wyświetlał półprzezroczysty rysunek.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 38.

1. **Otwórz w edytorze tekstu plik *5.2.html* z katalogu *R05*.**

   Pierwszy krok polega na dołączeniu pliku biblioteki jQuery.

2. **W pustym wierszu, tuż przed zamykającym nagłówkiem `</head>`, dodaj następujący kod:**

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

   Ten wiersz dołącza bibliotekę jQuery. Musisz go umieścić *przed* kodem JavaScript korzystającym z funkcji jQuery, dlatego warto wstawić go przed innymi znacznikami `<script>` na stronie.

   Następnie należy dodać własny kod JavaScript.

3. **Wciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz, i wpisz poniższy kod:**

   ```
   <script type="text/javascript">
   ```

   W tym momencie warto od razu zamknąć znacznik `<script>`.

4. **Wciśnij dwukrotnie klawisz *Enter* i wpisz `</script>`.**

   Teraz można użyć biblioteki jQuery.

5. **Dodaj do strony kod wyróżniony pogrubieniem:**

   ```
   <script type="text/javascript" src="../js/jquery.js"></script>
   <script type="text/javascript">
   $(document).ready(function() {

   });
   </script>
   ```

   Ten kod widziałeś już w przykładzie na stronie 42. Zastosowaną tu technikę poznasz szczegółowo w następnym rozdziale. Na razie zapamiętaj, że ten fragment gwarantuje wczytanie kodu HTML przed uruchomieniem programu JavaScript. Jest to bardzo ważne przy manipulowaniu stronami za pomocą skryptów JavaScript, ponieważ jeśli taki program spróbuje dodać, usunąć lub przekształcić kod HTML przed jego wczytaniem, wystąpi błąd.

   Teraz należy pobrać elementy za pomocą jQuery.

6. **Między dwoma dodanymi wierszami wpisz wyrażenie `$('#main h2')`. Kod powinien wyglądać następująco:**

   ```
   <script type="text/javascript" src="../js/jquery.js"></script>
   <script type="text/javascript">
   $(document).ready(function() {
     $('#main h2')
   });
   </script>
   ```

   Fragment `#main h2` to selektor potomków znany z języka CSS. Pobiera on wszystkie znaczniki `<h2>` z tagu o identyfikatorze `#main`, czyli każdy znacznik `<h2>` w głównej części strony. Następnie należy użyć pobranych znaczników.

7. **Wpisz fragment `.prepend('<span class="headEffect"></span>');`, aby kod wyglądał jak poniższy:**

   ```
   <script type="text/javascript" src="../js/jquery.js"></script>
   <script type="text/javascript">
   $(document).ready(function() {
   ```

```
  $('#main h2').prepend('<span class="headEffect"></span>');
});
</script>
```

Funkcja `.prepend()` dodaje kod tuż po otwierającym znaczniku pobranych elementów. Oznacza to, że skrypt doda pusty znacznik `<span>` wewnątrz każdego tagu `<h2>`, czyli przekształci kod HTML na przykład z `<h2>Tajemniczy nagłówek</h2>` na `<h2><span class="headEffect"></span> Tajemniczy nagłówek</h2>`.

8. **Zapisz stronę i wyświetl ją w przeglądarce.**

   Trzy duże, pogrubione, niebieskie nagłówki powinny wyglądać jak te z rysunku 5.8 (ukończony kod znajdziesz w pliku *kompletny_5.2.html*). Istotą tej techniki nie jest wykorzystanie kodu JavaScript lub biblioteki jQuery, ale stylów CSS. Każdy znacznik `<span>` dodany do nagłówków ma określony styl, który tworzy pole o wysokości 36 pikseli, umieszczane nad nagłówkami. Styl wyświetla w tym obszarze półprzezroczysty rysunek, co tworzy maskę, która ukrywa część tekstu nagłówków (zwróć uwagę na styl klasy `.headEffect`, zdefiniowany w wewnętrznym arkuszu umieszczonym w początkowej części dokumentu).

Możesz się zastanawiać, po co używać kodu JavaScript, skoro efekt jest wywoływany za pomocą języka CSS. Bez skryptu trzeba ręcznie dodać znaczniki `<span>` do każdego nagłówka, w którym programista chce zastosować dany efekt. Wymaga to wiele pracy, nie wspominając już o dodatkowym kodzie na stronie WWW. Ponadto jeśli efekt Ci się znudzi (czy to możliwe?), bez skryptu będziesz musiał znaleźć i usunąć niepotrzebne już znaczniki `<span>`. Przy zastosowanym podejściu wystarczy wykasować krótki fragment kodu JavaScript.

Obsługa wszystkich pobranych elementów

Na stronie 181 dowiedziałeś się, że jedną z wyjątkowych cech jQuery jest to, iż większość funkcji automatycznie przechodzi w pętli po wszystkich elementach pobranych za pomocą tej biblioteki. Aby stopniowo ukryć wszystkie elementy `<img>` na stronie, wystarczy użyć jednego wiersza kodu JavaScript:

```
$('img').fadeOut();
```

Funkcja `.fadeOut()` powoduje stopniowe zanikanie elementu, a jeśli użyjesz jej do kolekcji znaczników pobranych za pomocą jQuery, funkcja usunie z ekranu każdy z nich przy użyciu pętli. Przechodzenie w pętli po zbiorze znalezionych elementów i wykonywanie dla nich serii tych samych zadań to standardowa operacja. W jQuery można ją przeprowadzić za pomocą funkcji `.each()`.

Załóżmy, że chcesz wyświetlić listę wszystkich zewnętrznych odnośników ze strony w ramce z bibliografią na dole ekranu. Taką ramkę możesz nazwać „Inne witryny wymienione w artykule", a aby ją utworzyć, musisz wykonać następujące zadania:

1. **Pobrać wszystkie odnośniki prowadzące poza witrynę.**
2. **Ustalić atrybut `HREF` (adres URL) każdego z tych odnośników.**
3. **Dodać pobrane adresy URL do listy odnośników w ramce z bibliografią.**

JQuery nie udostępnia wbudowanej funkcji, która wykonuje te operacje, jednak można ją utworzyć samodzielnie przy użyciu funkcji `each()`. Jest to standardowa funkcja omawianej biblioteki, dlatego należy ją umieścić po instrukcji jQuery pobierającej kolekcję elementów:

```
$('selektor').each();
```

Funkcje anonimowe

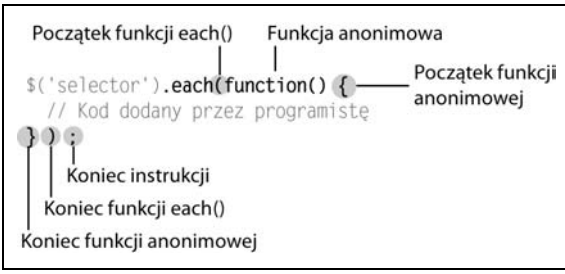
Aby użyć funkcji `each()`, należy przekazać do niej argument specjalnego rodzaju — *funkcję anonimową*. Jest to po prostu funkcja zawierająca operacje, które programista chce przeprowadzić na każdym z pobranych elementów. Określenie *anonimowa* wynika z tego, że takie funkcje — w odróżnieniu od tych, które poznałeś na stronie 105 — nie mają nazwy. Oto podstawowa struktura funkcji anonimowej:

```
function() {
  // Tu uruchamiany kod.
}
```

Ponieważ taka funkcja nie ma nazwy, nie można jej wywołać. W zamian należy przekazać ją jako argument do innej funkcji (dziwne, ale prawdziwe!). W poniższym kodzie funkcja anonimowa jest częścią funkcji `each()`:

```
$('selektor').each(function() {
  // Tu uruchamiany kod.
});
```

Na rysunku 5.9 opisano różne części tej struktury. Szczególnie zawiły jest ostatni wiersz, który zawiera trzy różne symbole zamykające trzy fragmenty całej instrukcji. Znak } kończy funkcję anonimową (jest to też koniec argumentu przekazanego do funkcji `each()`), nawias) to ostatni symbol funkcji `each()`, a znak ; kończy instrukcję języka JavaScript. Oznacza to, że interpreter traktuje cały podany kod jak jedną instrukcję.



Rysunek 5.9. *Funkcja each() biblioteki jQuery pozwala przejść w pętli po kolekcji pobranych elementów i wykonać dla nich serię zadań. Istotą tej techniki są funkcje anonimowe*

Po przygotowaniu struktury zewnętrznej pora dodać kod do funkcji anonimowej, czyli operacje przeprowadzane na wszystkich pobranych elementach. Funkcja `each()` działa jak pętla, co oznacza, że skrypt uruchomi instrukcje z funkcji anonimowej dla każdego pobranego znacznika. Załóżmy, że na stronie znajduje się 50 rysunków, a jeden ze skryptów zawiera następujący kod:

```
$('img').each(function() {
  alert('Znaleziono rysunek');
});
```

Pojawi się 50 okien dialogowych z informacją „Znaleziono rysunek". To naprawdę irytujące, dlatego nie używaj tego kodu we własnych witrynach.

Słowo kluczowe this i konstrukcja $(this)

W czasie korzystania z funkcji `each()` programista oczywiście chce móc pobierać i ustawiać atrybuty poszczególnych elementów, na przykład aby ustalić adres URL każdego odnośnika zewnętrznego. Dostęp do aktualnie przetwarzanego elementu zapewnia słowo kluczowe `this`. Wskazuje ono element, dla którego wywołano funkcję anonimową. Dlatego przy pierwszym uruchomieniu pętli słowo `this` odpowiada pierwszemu znacznikowi kolekcji pobranej za pomocą jQuery, przy drugim uruchomieniu — elementowi drugiemu i tak dalej.

Słowo kluczowe `this` oznacza tradycyjny element modelu DOM, dlatego daje dostęp do standardowych właściwości tego modelu, na przykład `innerHTML` (strona 165) i `childNodes` (strona 163). Jednak, jak opisano we wcześniejszej części rozdziału, elementy pobrane za pomocą jQuery umożliwiają korzystanie ze wszystkich wygodnych funkcji tej biblioteki. Dlatego aby przekształcić słowo kluczowe `this` na element biblioteki jQuery, należy użyć konstrukcji `$(this)`.

Korzystanie ze słowa kluczowego `this` może być skomplikowane. Aby lepiej zrozumieć działanie konstrukcji `$(this)`, zastanów się ponownie nad zadaniem opisanym na początku punktu — tworzeniem listy odnośników zewnętrznych w ramce z bibliografią na dole strony.

Załóżmy, że strona zawiera już znacznik `<div>`, przeznaczony na odnośniki zewnętrzne:

```
<div id="bibliography">
<h3>Strony wymienione w artykule</h3>
<ul id="bibList">
</ul>
</div>
```

Pierwszy krok polega na pobraniu listy wszystkich odnośników prowadzących poza witrynę. Można je znaleźć za pomocą selektora atrybutów (strona 178):

```
$('a[href^=http://]')
```

Aby przejść w pętli po wszystkich odnośnikach, można użyć funkcji `each()`:

```
$('a[href^=http://]').each()
```

Następnie należy dodać funkcję anonimową:

```
$('a[href^=http://]').each(function() {

});
```

Pierwszy krok w funkcji anonimowej to pobranie adresu URL odnośnika. Ponieważ każdy odsyłacz zawiera inny adres, przy każdym powtórzeniu pętli należy uzyskać dostęp do aktualnie przetwarzanego elementu. Umożliwia to konstrukcja `$(this)`:

```
$('a[href^=http://]').each(function() {
  var extLink = $(this).attr('href');
});
```

Kod wyróżniony pogrubieniem wykonuje kilka operacji. Po pierwsze, tworzy nową zmienną (`extLink`) i zapisuje w niej wartość właściwości `href` aktualnego elementu. Przy każdym powtórzeniu pętli konstrukcja `$(this)` odpowiada innemu odnośnikowi, dlatego zmienna `extLink` za każdym razem ma odmienną wartość.

Następnie wystarczy dodać nowy element listy do znacznika `<ul>` (zobacz kod HTML na stronie 194):

```
$('a[href^=http://]').each(function() {
  var extLink = $(this).attr('href');
  $('#bibList').append('<li>' + extLink + '</li>');
});
```

Prawie zawsze kiedy będziesz używał funkcji `each()`, będziesz stosował także konstrukcję `$(this)`, dlatego szybko się do niej przyzwyczaisz. Aby przećwiczyć tę technikę, wykorzystasz ją w następnym przykładzie.

Uwaga: Przykładowy skrypt z tego punktu dobrze ilustruje używanie konstrukcji `$(this)`, jednak nie przedstawia optymalnej metody wyświetlania na stronie listy zewnętrznych odnośników. Po pierwsze, jeśli strona nie będzie zawierać takich odsyłaczy, widoczny będzie na niej pusty znacznik `<div>`, zapisany na stałe w kodzie HTML. Ponadto jeżeli użytkownik otworzy stronę za pomocą przeglądarki bez włączonej obsługi języka JavaScript, nie zobaczy odnośników, a tylko pustą ramkę. Lepsze rozwiązanie polega na użyciu języka JavaScript także do utworzenia znacznika `<div>`. Wśród przykładowych plików znajdziesz stronę *bibliography.html*, na której zastosowano to podejście.

Automatyczne ramki z cytatami

W ostatnim przykładzie w tym rozdziale utworzysz skrypt, który ułatwia dodawanie do strony *ramek z cytatami* (rysunek 5.10). Taka ramka to pole zawierające ciekawy cytat z głównego tekstu. W dziennikach, magazynach i witrynach te obszary mają przyciągać uwagę czytelników i podkreślać ważne lub ciekawe fragmenty artykułów. Jednak ręczne dodawanie ramek z cytatami wymaga powielania tekstu i umieszczania go w znaczniku `<div>` lub `<span>` albo w innym kontenerze. To podejście zajmuje dużo czasu, a ponadto powoduje umieszczenie na gotowej stronie dodatkowego kodu HTML i skopiowanego tekstu. Na szczęście przy użyciu kodu JavaScript można szybko dodać do strony dowolną liczbę ramek z cytatami, używając małej ilości kodu HTML.

Omówienie przykładu

Nowy skrypt ma wykonywać kilka zadań. Są to:

1. **Znajdowanie wszystkich znaczników `<span>` klasy `pq`.**

   W kodzie HTML strony wystarczy dodać znaczniki `<span>` wokół tekstu używanego w ramkach z cytatami. Załóżmy, że chcesz wyświetlić w takiej ramce kilka słów z akapitu tekstu. Wystarczy umieścić dany fragment w znacznikach `<span>`:

   ```
   <span class="pq">...i właśnie w ten sposób odkryłem potwora z Loch
   ↪Ness.</span>
   ```

Rysunek 5.10. *Ręczne dodawanie ramek z cudzysłowami do stron HTML jest męczące, zwłaszcza że można zautomatyzować ten proces za pomocą języka JavaScript*

2. **Powielanie każdego znacznika `<span>`.**

   Każda ramka z cytatem to nowy znacznik `<span>` z pierwotnym tekstem, dlatego można użyć kodu JavaScript do skopiowania oryginalnego tagu.

3. **Usuwanie klasy `pq` z powielonego znacznika `<span>` i dodawanie do niego klasy `pullquote`.**

   Magia formatowania, która powoduje utworzenie ramki z cytatem — pole, większy tekst, obramowanie i kolor tła — to nie efekt działania kodu JavaScript. Arkusz stylów strony zawiera selektor klasy CSS, `.pullquote`, w którym określono całe formatowanie. Dlatego przez zmianę nazwy klas w skopiowanym znaczniku za pomocą kodu JavaScript można całkowicie zmodyfikować wygląd nowych znaczników `<span>`.

4. **Dodawanie skopiowanych znaczników `<span>` do strony.**

   Na zakończenie należy dodać powielony znacznik `<span>` do strony. W kroku 2. skrypt tworzy kopię tagu w pamięci przeglądarki, ale nie umieszcza go na stronie. Daje to dodatkową możliwość zmodyfikowania skopiowanego znacznika przed wyświetleniem go użytkownikowi.

Tworzenie kodu

Wiesz już, jakie zadania ma wykonywać skrypt. Pora otworzyć edytor tekstu i przygotować rozwiązanie.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 38.

1. **Otwórz w edytorze tekstu plik *5.3.html* z katalogu *R05*.**

   Strona zawiera już kod dołączający plik *jquery.js*, a także znacznik `<script>` z dziwnym fragmentem `$(document).ready`, który napotkałeś w kroku 5. na stronie 191.

2. **Kliknij pusty wiersz pod fragmentem `$(document).ready` i dodaj kod wyróżniony pogrubieniem:**

```
1 <script type="text/javascript" src="../js/jquery.js"></script>
2 <script type="text/javascript">
3 $(document).ready(function() {
4   $('span.pq')
5
6 });
7 </script>
```

Uwaga: Numery po lewej stronie każdego wiersza są podane tylko w celu ułatwienia opisu kodu. Pomiń je w czasie wpisywania skryptu na stronie.

Fragment `$('span.pq')` to selektor jQuery, który pobiera wszystkie znaczniki `<span>` klasy `pq`. Następnie należy dodać kod, który przejdzie w pętli po każdym z tych znaczników i wykona na nich określone operacje.

3. **Dodaj kod wyróżniony pogrubieniem (wiersze od 4. do 6.):**

```
1 <script type="text/javascript" src="../js/jquery.js"></script>
2 <script type="text/javascript">
3 $(document).ready(function() {
4   $('span.pq').each(function() {
5
6   });
7 });
8 </script>
```

Na stronie 193 dowiedziałeś się, że funkcja `.each()` biblioteki jQuery umożliwia przejście w pętli po pobranych elementach. Funkcja ta przyjmuje jeden argument, który sam jest funkcją.

Następnie należy utworzyć funkcję uruchamianą dla każdego pobranego znacznika `<span>`. Pierwsza operacja polega na utworzeniu kopii tego elementu.

4. **Dodaj do skryptu kod wyróżniony pogrubieniem (wiersz 5.):**

```
1 <script type="text/javascript" src="../js/jquery.js"></script>
2 <script type="text/javascript">
3 $(document).ready(function() {
4   $('span.pq').each(function() {
5     var quote=$(this).clone();
6   });
7 });
8 </script>
```

Funkcja najpierw tworzy nową zmienną, `quote`, i zapisuje w niej „klon" (kopię) aktualnie przetwarzanego znacznika `<span>` (opis konstrukcji `$(this)` znajdziesz na stronie 194). Funkcja `.clone()` biblioteki jQuery powiela element, w tym cały umieszczony w nim kod HTML. Tu funkcja ta tworzy kopię znacznika `<span>` i zapisanego w nim tekstu, który pojawi się w ramce z cytatem.

Klon elementu to pełna kopia, włącznie z atrybutami. W przykładowym kodzie pierwotny znacznik `<span>` ma klasę `pq`, którą należy usunąć z kopii.

5. **Dodaj w dolnej części skryptu dwa wiersze kodu wyróżnione pogrubieniem (wiersze 6. i 7.):**

```
1  <script type="text/javascript" src="../js/jquery.js"></script>
2  <script type="text/javascript">
3  $(document).ready(function() {
4    $('span.pq').each(function() {
5      var quote=$(this).clone();
6      quote.removeClass('pq');
7      quote.addClass('pullquote');
8    });
9  });
10 </script>
```

Na stronie 186 dowiedziałeś się, że funkcja `removeClass()` usuwa klasę ze znacznika, natomiast funkcja `.addClass()` dołącza nazwę klasy. Dodane wiersze zastępują pierwotną klasę w kopii elementu, dlatego można użyć klasy CSS `.pullquote` do przekształcenia znaczników `<span>` na ramki z cytatami.

Teraz można dodać znacznik `<span>` do strony.

6. **Dodaj do skryptu kod wyróżniony pogrubieniem (wiersz 8.):**

```
1  <script type="text/javascript" src="../js/jquery.js"></script>
2  <script type="text/javascript">
3  $(document).ready(function() {
4    $('span.pq').each(function() {
5      var quote=$(this).clone();
6      quote.removeClass('pq');
7      quote.addClass('pullquote');
8      $(this).before(quote);
9    });
10  });
11 </script>
```

Ten wiersz to ostatni fragment funkcji. Do tego miejsca skrypt jedynie manipulował kopią znacznika `<span>` w pamięci przeglądarki. Użytkownik strony nie zobaczy tej kopii do momentu dodania jej do modelu DOM.

Użyty kod wstawia kopię znacznika `<span>` tuż przed jego pierwotną wersją. Na stronie pojawi się kod HTML podobny do poniższego:

```
<span class="pullquote">...i właśnie w ten sposób odkryłem potwora
↪z Loch Ness.
</span> <span class="pq">...i właśnie w ten sposób odkryłem potwora
↪z Loch
Ness.</span>
```

Choć może się wydawać, że tekst oryginalny i pierwotny pojawią się na stronie obok siebie, styl CSS powoduje przesunięcie ramki z cytatem do prawej krawędzi okna.

Kod JavaScript jest już gotowy. Jednak nie zobaczysz ramek z cytatami, jeśli nie zmodyfikujesz nieco kodu HTML.

Uwaga: Aby uzyskać efekt wizualny ramki z cytatem, w stylu CSS użyto właściwości `float`. Ramka jest przesuwana do prawej krawędzi akapitu, w którym pojawia się oryginalny fragment, a pozostały tekst opływa ramkę. Jeśli chcesz poznać tę technikę, na stronie *http://css.maxdesign.com.au/floatutorial/* znajdziesz informacje o właściwości `float`.

7. **Znajdź pierwszy znacznik `<p>` w kodzie HTML strony. Wybierz zdanie i zapisz je wewnątrz znaczników `<span class="pq"></span>`, na przykład:**

```
<span class="pq">Nullam ut nibh sed orci tempor rutrum.</span>
```

   Możesz powtórzyć ten proces, aby dodać ramki z cytatami także w innych akapitach.

8. **Zapisz plik i wyświetl go w przeglądarce.**

   Ostateczny efekt powinien wyglądać jak na rysunku 5.10. Jeśli nie widzisz ramek z cytatami, sprawdź, czy w kroku 7. prawidłowo dodałeś znacznik `<span>`. Ponadto zapoznaj się ze wskazówkami ze strony 44, które są pomocne przy naprawianiu błędów w programach. Gotową wersję kodu z tego przykładu znajdziesz w pliku *kompletny_5.3.html*.