Ajax. Implementacje

Autor: Shelley Powers
Tłumaczenie: Tomasz Walczak
ISBN: 978-83-246-1330-4
Tytuł oryginału: Adding Ajax
Format: B5, stron: 360

Podążaj za swoją wyobraźnią
i usprawniaj strony www za pomocą efektów Ajaksa!

- Jak używać kodu JavaScript do rozwiązywania problemu niestandardowych atrybutów?
- Jak łączyć funkcje obsługi zdarzeń?
- Jak tworzyć zaawansowane efekty bazujące na CSS?

Ajax to coś więcej niż zbiór technologii obejmujący języki oparte na znacznikach, jest bowiem narzędziem, które – ewoluując – na bieżąco dotrzymuje kroku rozwijającej się technice informatycznej, a także wyobraźni programistów. Po zastosowaniu stopniowego usprawniania można dodawać nowoczesne efekty Ajaksa i w ten sposób przenosić tradycyjne aplikacje sieciowe i ich funkcjonalność na wyższy poziom.

„Ajax. Implementacje" to książka dla programistów, którzy mają doświadczenie w tworzeniu witryn internetowych i są zainteresowani ulepszaniem istniejących już aplikacji. Czytając ją, nauczysz się, jak zwiększać możliwości stron www poprzez dodawanie do nich efektów Ajaxa, dowiesz się, jak korzystać z języka SVG oraz obiektu Canvas. Poznasz jedną z najciekawszych możliwości Ajaksa, czyli dodawanie usług sieciowych i łączenie danych bezpośrednio na stronach internetowych na wiele różnych sposobów. „Ajax. Implementacje" zawiera całą niezbędną wiedzę potrzebną, aby mieć nowoczesne i funkcjonalne strony www, nie budując ich od nowa.

- Stronicowanie w Ajaksie
- Strefa bezpieczeństwa i zabezpieczenia języka JavaScript
- Obsługa zdarzeń zgodna z Ajaksem
- System obsługi zdarzeń Dojo i obiekty docelowe
- Dane dynamiczne
- Prawidłowe współdziałanie Ajaksa z innymi elementami aplikacji
- Efekty bibliotek zewnętrznych służące do obsługi danych
- Historia, nawigacja i miejsca w aplikacjach jednostronicowych
- Dodawanie zaawansowanych efektów wizualnych
- Witryny typu mashup
- Skalowanie, infrastruktura i tworzenie witryn od podstaw

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

Spis treści

ROZDZIAŁ 8.

Dodawanie zaawansowanych efektów wizualnych

Ajax nie wymaga skomplikowanych efektów wizualnych. Jedyne potrzebne efekty to te przedstawione w poprzednich rozdziałach: możliwość dodawania, zmieniania i usuwania zawartości strony, ukrywania i wyświetlania nowych danych oraz wyróżniania zmian. Wszystkie te efekty opierają się na stosunkowo prostych i rozpowszechnionych technologiach, takich jak HTML czy CSS. Po drugiej stronie rodziny aplikacji ajaksowych znajdują się bogate aplikacje internetowe (ang. *Rich Internet Applications* — RIA). Takie programy mają naśladować w przeglądarkach internetowych tradycyjne aplikacje stacjonarne, a do ich tworzenia programiści używają technologii ajaksowych. Aplikacje tego typu to edytory tekstu, arkusze kalkulacyjne, klienty poczty elektronicznej, a nawet programy graficzne, podobne do Painta, GIMP-a (ang. *GNU Image Manipulation Program*) czy Photoshopa.

Aplikacje RIA stawiają większe wymagania funkcjom graficznym przeglądarek. Tych potrzeb zwykle nie zaspokajają style CSS, a na pewno nie ich prostsze zastosowania. Choć tworzenie aplikacji tego typu znacznie wykracza poza zakres tej książki, inaczej jest w przypadku narzędzi wizualnych używanych przez autorów programów RIA. A przynajmniej nie wykraczają one na tyle daleko, żeby nie poświęcić jednego rozdziału na odrobinę zabawy i nie spędzić nieco czasu na poznawaniu efektów, jakie można dodać do edycji w miejscu czy sprawdzania aktualizacji.

Przy dodawaniu zaawansowanych efektów wizualnych do aplikacji sieciowych można zastosować dwa podejścia. Pierwsze z nich jest dostępne we wszystkich przeglądarkach internetowych i opiera się na efektach zarządzanych za pomocą stylów CSS. Obejmuje to przezroczystość, zmiany rozmiaru, przycinanie i tak dalej.

Drugie podejście polega na wykorzystaniu wyspecjalizowanych obiektów i specyfikacji. W tym przypadku programiści mają szczęście w nieszczęściu, ponieważ dostępnych jest wiele różnych bibliotek graficznych, obiektów i technologii, spośród których można wybierać. Duży wybór jest korzystny, a dotyczy to także programowania aplikacji sieciowych, o ile dostępne narzędzia działają we wszystkich docelowych przeglądarkach. I tu leży problem związany ze specyfikacjami efektów wizualnych dostępnymi programistom Ajaksa: obsługa zaawansowanych efektów wizualnych jest wbudowana w niemal wszystkie powszechnie używane przeglądarki, jednak każda z nich współdziała tylko z niektórymi rozwiązaniami, a żadna technika nie jest powszechnie obsługiwana we wszystkich najważniejszych przeglądarkach.

Inne firmy promują następną specyfikację graficzną: skalowalną grafikę wektorową (ang. *Scalable Vector Graphics* — SVG). Format SVG jest obecnie obsługiwany w przeglądarkach opartych na Gecko (takich, jak: Mozilla, Camino, Firefox), w Operze, a także w rozwojowej wersji Safari (WebKit). Dostępne są wtyczki, które zapewniają obsługę formatu SVG także w innych przeglądarkach, między innymi w Internet Explorerze. Specyfikację SVG zatwierdziło konsorcjum W3C i prawdopodobnie w przyszłości format ten stanie się standardem.

Firma Apple utworzyła rozszerzenie Canvas dla przeglądarki WebKit. Udostępnia ono implementację grafiki dla widgetów Dashboard i przeglądarki Safari. Zapewnia dostęp do interfejsu API Cocoa 2D z poziomu języka JavaScript, co ma znaczenie oczywiście wyłącznie w systemie Mac OS X. Mimo to element `canvas` został uwzględniony w standardzie Web Applications 1.0 HTML firmy WhatWG, nazywanym powszechnie specyfikacją HTML 5. Dlatego obsługa tego elementu znajduje się także w przeglądarkach opartych na Gecko oraz w Operze i Safari.

Ponieważ obsługa formatu VML jest ograniczona, nie omawiam go w tym rozdziale. Wspominam o nim, tylko opisując biblioteki, które zapewniają obsługę VML i SVG w różnych przeglądarkach. W zamian koncentruję się na elemencie `canvas` i formacie SVG oraz przedstawiam bardziej zaawansowane zastosowania CSS.

Prace nad HTML5 w WhatWG zostały skoordynowane z W3C w celu utworzenia nowej, zaktualizowanej wersji języka HTML. Te wysiłki koncentrują się na udostępnieniu nowej, ustandaryzowanej wersji znaczników, będącej alternatywą dla programistów witryn internetowych, którzy nie są gotowi do przejścia na XHTML. Więcej informacji o specyfikacji HTML5 WhatWG można znaleźć na stronie *http://www.whatwg.org/specs/web-apss/current-work* oraz w witrynie W3C pod adresem *http://www.w3.org/html/wg*.

Zaawansowane sztuczki z CSS

Zawsze uważałam, że efekty wizualne przedstawione we wcześniejszych rozdziałach — takie, jak zapewnianie przezroczystości obiektów, ukrywanie i wyświetlanie elementów czy przenoszenie ich na żądanie — są dość zaawansowane. Jednak w aplikacjach ajaksowych można zastosować także inne sztuczki oparte na CSS.

Czy którakolwiek ze sztuczek prezentowanych w tym miejscu jest niezbędna? Moim zdaniem żadne efekty nie są konieczne — i nie powinny być, jeśli jednym z celów jest to, aby zastosowanie Ajaksa było dyskretne. Jednak efekty pozwalają dodać elegancji stronom internetowym, co może być równie ważne jak zapewnienie dostępności. Co jednak najlepsze, jeśli efekty są już wbudowane w bibliotekę, można oszczędzić sobie wiele pracy.

Zaokrąglone narożniki

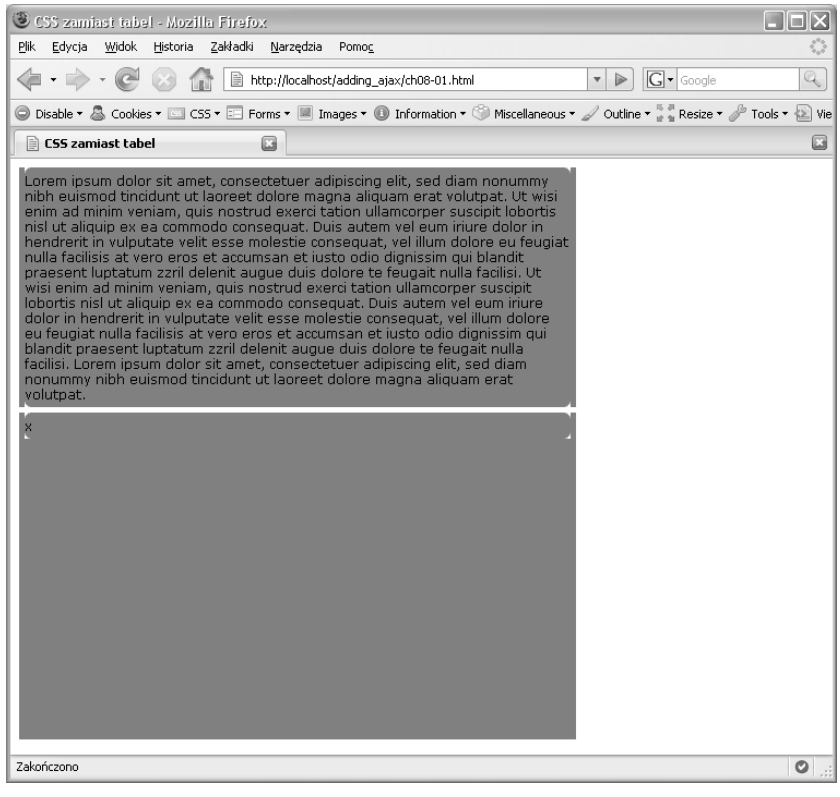
Ajax i zaokrąglone narożniki pasują do siebie jak ręka do rękawiczki. Zwykle do utworzenia tego efektu używane są rysunki zaokrąglonych narożników wyświetlane jak tło dla wielu warstw, co tworzy „pudełko", którego rozmiar można zmienić w pionie i poziomie.

Inne rozwiązania, oparte na obrazach PNG i przezroczystości, pozwalają „tworzyć" zaokrąglone narożniki po kilka pikseli naraz. Nie przepadam za tą techniką, zwłaszcza ze względu na słabą obsługę przezroczystości alfa obrazów PNG w przeglądarce Internet Explorer.

Kilka omówionych w tej książce bibliotek Ajaksa udostępnia zaokrąglone narożniki tworzone przy użyciu kodu JavaScript. Rozwiązanie z biblioteki Rico jest proste w użyciu i udostępnia opcje umożliwiające przenikanie oraz określanie tylko jednego narożnika, na przykład opcję `tl` (górny lewy narożnik):

```
Rico.Corner.round('div1'); // Określenie identyfikatora elementu
Rico.Corner.round('div', {corners: "tl"});
```

Proste, jednak zaokrąglanie za pomocą Rico ma specyficzny efekt uboczny. Jeśli programista użyje dopełnienia lub ustawi wysokość elementu na większą niż długość treści, aplikacja wyświetli „powygryzany" blok, co przedstawia rysunek 8.1.



Rysunek 8.1. Dziwny efekt zaokrąglonych narożników

Sposobem na uniknięcie „wcięć" jest rezygnacja z dopełnienia i nieustawianie wysokości. Jeśli programista chce zastosować wewnętrzne dopełnienie, może użyć bloku wewnętrznego i odpowiednio ustawić jego marginesy. Narożniki z biblioteki Rico będą wtedy działały prawidłowo. Więcej przykładów zaokrąglania za pomocą Rico można zobaczyć na stronie *http://openrico.org/demos/?demo=corner*.

Pakiet MochiKit także obsługuje zaokrąglone narożniki. Służy do tego biblioteka Visual. Umożliwia ona zaokrąglanie bloków całej klasy elementów, a nie tylko po jednym naraz:

```
roundClass('div');
```

Jest to wygodna opcja. Jednak aby jej użyć, trzeba dołączyć kilka bibliotek:

```
MochiKit/Base.js
MochiKit/Iter.js
MochiKit/DOM.js
MochiKit/Color.js
MochiKit/Visual.js
```

Jednak MochiKit to kompletna biblioteka Ajaksa, a także infrastruktura. Jeśli programista używa innych komponentów do odmiennych zadań, może zastosować bibliotekę Visual do zaokrąglania. W przeciwnym razie warto zrezygnować z krzywizn. MochiKit, podobnie jak Rico, także zagrożona jest efektem „wcięć" i trzeba użyć wewnętrznych elementów z marginesem, jeśli potrzebne są dopełnienia. MochiKit udostępnia bardzo ciekawą dokumentację, o czym można się przekonać na przykładzie biblioteki Visual.js, odwiedzając stronę *http://mochikit.com/doc/html/MochiKit/Visual.html*.

Zaokrąglone narożniki mogą dodać stronie internetowej elegancji. Decyzja o użyciu w tym celu grupy rysunków lub kodu JavaScript zależy od tego, czy programista używa już biblioteki. Warto jednak pamiętać o tym, że jeśli rozwiązanie opiera się na kodzie JavaScript, a obsługa skryptów jest wyłączona, efekt zostanie utracony.

Suwaki i paski przewijania

Suwaki to elementy wizualne składające się z uchwytu, dwóch punktów końcowych oraz wąskiego prostokąta o szerokości uchwytu, ograniczonego przez punkty końcowe. Już niektóre z pierwszych dynamicznych pakietów z komponentami wizualnymi HTML miały wbudowaną obsługę pasków przewijania, a obecnie można znaleźć mnóstwo bibliotek języka JavaScript i wyspecjalizowanych bibliotek Ajaksa zapewniających tę funkcjonalność. Wszystkie powinny zapewniać obsługę zdarzeń, a także dostęp za pomocą klawiatury i alternatywne, tekstowe rozwiązanie w przypadku wyłączonej obsługi skryptów. Powinny, ale w wielu nie ma alternatywy dla skryptów — zwykle trzeba zapewnić ją samemu.

Paski przewijania to odmiana suwaków. Znajdują się w prawej lub dolnej części elementu i służą do przewijania w pionie lub poziomie treści, która wykracza poza ramy danego elementu.

Paski przewijania można wbudować w dowolny element, używając stylów CSS. Ustawienie właściwości `overflow` na `scroll` powoduje automatyczne dodanie pasków przewijania do elementu. Niestety, takie paski są widoczne nawet wtedy, kiedy materiał nie wykracza poza obszar elementu, jednak poprawne używanie tej techniki może nadać elementom nowy wymiar. Inne rozwiązanie polega na nadaniu określonej wysokości elementowi i ustawieniu `overflow` na wartość `auto`, co powoduje, że pasek przewijania będzie widoczny tylko wtedy, kiedy zawartość przekroczy rozmiar elementu.

Ajaksowe paski przewijania zwykle wiążą się z zagadnieniem ajaksowego stronicowania, choć samych suwaków można używać do różnych zadań, włączając w to dostosowywanie kolorów obiektów i zmianę szerokości elementów.

Jest tak wiele dobrych bibliotek do tworzenia suwaków i pasków przewijania, że nie mogę wyobrazić sobie, aby potrzebne było tworzenie własnych. Yahoo! UI ma wbudowany suwak, a w X Library Mike'a Fostera także znajduje się ciekawe rozwiązanie (ta biblioteka oraz jej dokumentacja są dostępne na stronie *http://cross-browser.com*). Dojo zawiera komponent w postaci suwaka, podobnie jak script.aculo.us. Wpisanie w wyszukiwarce słowa „slider" prawdopodobnie pozwoli znaleźć komponent w postaci suwaka, dokumentację stosowania wbu-

dowanych suwaków i przykłady ich zastosowania. A są to tylko zastosowania ajaksowe. Po wpisaniu wyrażenia „javascript and slider" można znaleźć bardziej zaawansowane wersje suwaków, które działają niezależnie od bibliotek Ajaksa.

Jak wspomniałam w poprzednich rozdziałach, bardzo cenię stronicowanie. Uważam, że nawigacja po zawartości strony podzielonej na warstwy w formie stron z zakładkami czy pokazu slajdów działa bardzo dobrze. Przy zwracaniu wyników z zapytania, jeśli zwrócone dane są proste, stronicowanie pozwala udostępnić ciekawy efekt. Niewątpliwie jest to dobry przykład zastosowania suwaków i pasków przewijania.

Następny przykład obejmuje stronicowanie przy użyciu usługi kierującej zapytania o dane. Usługa ta została utworzona w rozdziale 7. na potrzeby jednostronicowej aplikacji (pokazu slajdów) korzystającej z bazy danych. Usługa działająca na zapleczu nie wymaga żadnych zmian. Fronton składa się z suwaka i elementu strony zawierającego tabelę HTML, w której wraz z przesuwaniem suwaka w dół wyświetlane są wiersze o coraz wyższych numerach. Tabela umożliwia przewijanie w górę i w dół na podstawie ruchów suwaka, dzięki czemu można wyświetlić niedawno pobrane lub starsze dane.

Poniższa strona internetowa jest bardzo prosta. Zawiera suwak oraz elementy przeznaczone na tabelę HTML (listing 8.1).

Listing 8.1. Przykładowa strona internetowa z suwakiem i stronicowaniem

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pl" xml:lang="pl">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Suwak</title>
<link rel="stylesheet" type="text/css" href="pagination.css" />
<!-- Pliki źródłowe z przestrzenią nazw -->
  <script src = "yui/build/yahoo/yahoo.js" ></script>

  <!-- Zależne pliki źródłowe -->
  <script type="text/javascript" src = "yui/build/dom/dom.js" ></script>
  <script type="text/javascript" src = "yui/build/event/event.js" ></script>
  <script type="text/javascript" src = "yui/build/dragdrop/dragdrop.js" ></script>

  <!-- Plik źródłowy do obsługi suwaka -->
  <script type="text/javascript" src = "yui/build/slider/slider.js" ></script>

  <!-- Pliki specyficzne dla aplikacji -->
  <script type="text/javascript" src="addingajax.js">
  </script>
  <script type="text/javascript" src="paginate.js">
  </script>

</head>
<body>
<div id="container">
<div id="sliderbg">
  <div id="sliderthumb"><img src="verticalSlider.gif" alt="suwak" /></div>
</div>
<div id="main">
<div id="inner">
</div>
</div>
</div>
</body>
</html>
```

Użyta miniatura pochodzi z przykładu z biblioteki YUI, jednak można zastosować dowolny inny rysunek. Plik CSS używany w tym przykładzie także jest prosty. Najważniejsze jest ustawienie pionowego suwaka po lewej stronie, obszaru z tabelą po prawej, a głównego obszaru tak, aby aplikacja ukrywała nadmiarowe wiersze w celu utworzenia efektu „przewijania" przy przenoszeniu danych na podstawie ruchów suwaka:

```
#container
{
     margin: 20px auto;
     width: 700px;
     height: 440px;
}
#sliderbg
{
     background-color: #ccc;
     border: 1px outset #333;
     float: left;
     height: 420px;
     width: 10px;
}
#main
{
     position: absolute;
     background-color: #ccc;
     border: 1px outset #333;
     height: 420px;
     margin-left: 20px;
     overflow: hidden;
     width: 600px;
}
#inner
{
     margin: 5px;
     top: 10px;
}
tr
{
     margin-bottom: 3px;
     margin-top: 3px;
}
td
{
     margin: 2px;
}
table
{
     width: 98%;
}
```

Kod JavaScript aplikacji, przedstawiony na listingu 8.2, to zmodyfikowana wersja kodu z listingu 7.9 z rozdziału 7. Jedyna różnica polega na tym, że w czasie wczytywania strony tworzona jest tabela i powiązany z nią suwak, a pobierane dane aplikacja umieszcza w tej właśnie tabeli.

Listing 8.2. Kod JavaScript do obsługi suwaka i stronicowania

```
var cache = {
   offset : 80,
   fetch : 40
};
```

```
aaManageEvent(window,"load", function() {
  var slider = YAHOO.widget.Slider.getVertSlider("sliderbg",
         "sliderthumb", 0, 400);
  slider.setValue(0,true);
  slider.subscribe("change",adjustPage);

  // Dostosowanie pozycji z uwagi na specyfikę przeglądarki Internet Explorer
  createTable();
  getRows(0);
  aaElem('inner').style.position='relative';
});

// Dostosowuje kontener na tabelę na podstawie pozycji
// i określa, czy trzeba zwiększyć pamięć podręczną
// dla bazy danych
function adjustPage(offset) {

   var inner = aaElem('inner');
   var newTop = -offset;
   inner.style.top = newTop + "px";

   // Jeśli przesunięcie jest większe niż w pamięci podręcznej,
   // należy pobrać wiersze i zaktualizować tę pamięć
   if (offset > cache.offset) {
      getRows(cache.fetch);
      cache.offset+=80;
      cache.fetch+=40;
   }
}

// Pobiera następną grupę wierszy
function getRows(start) {
   var url = 'getposts.php?start=' + start;
   var script = document.createElement('script');
   script.type = 'text/javascript';
   script.src = url + '&limit=40&callback=printRows';
   document.getElementsByTagName('head')[0].appendChild(script);
}

// Tworzy i dołącza komórkę tabeli
function createTableCell(value,tr) {

   // Tworzy element <td>
   var cell = document.createElement("td");

   // Tworzy węzeł tekstowy
   var text = document.createTextNode(value);

   // Dołącza utworzony węzeł tekstowy do komórki <td>
   cell.appendChild(text);

   // Dołącza komórkę <td> do wiersza <tr>
   tr.appendChild(cell);

   return tr;
}

// Tworzy pustą tabelę i dodaje do strony;
// ustawia styl top kontenera
function createTable() {
   var inner = aaElem('inner');

   var table = document.createElement('table');
```

```
    table.id = 'dataTable';
    var tableBody = document.createElement("tbody");
    tableBody.id = "dataTableBody";
    table.appendChild(tableBody);
    inner.appendChild(table);
    inner.style.top = "0px";
}

// Dodaje nowo pobrane wiersze jako wiersze tabeli
function printRows(rowsObj) {

    // Usuwa stare dane
    var tableBody = aaElem('dataTableBody');

    // Wiersze tabeli
    for(var i = 0; i < rowsObj.length; i++) {
        // Tworzy element <tr>
        var row = document.createElement("tr");

        row = createTableCell(rowsObj[i].id, row);

        // Tworzy znacznik anchor i tytuł
        var cell = document.createElement("td");
        cell.innerHTML = "<a href='" + rowsObj[i].guid + "'>" + rowsObj[i].title +
                         "</a>";
        row.appendChild(cell);

        row = createTableCell(rowsObj[i].comments,row);

       tableBody.appendChild(row);
       tableBody.style.overflow="hidden";
    }
}
```

Po poruszeniu suwaka atrybut `top` stylów CSS tabeli jest ustawiany na wartość ujemną przesunięcia suwaka (jego odległości od góry). Dzięki temu tabela zawsze jest przewijana w górę wraz z przeciąganiem suwaka w dół, a aplikacja przewija ją w dół, kiedy użytkownik przeciągnie suwak w górę. Na początku pobieranych jest jedynie 40 wierszy, a globalny obiekt służący jako pamięć podręczna służy do określania liczby wczytanych wierszy. W tym celu aplikacja sprawdza, czy przesunięcie suwaka jest większe niż wartość ustawiona przy ostatniej aktualizacji pamięci podręcznej. Pasek przewijania o długości 400 pikseli wymaga w tym przypadku mniej więcej pięciu operacji pobierania danych z bazy.

Układ jest domyślnie statyczny, a aby przycinanie działało prawidłowo, wewnętrznemu elementowi zawierającemu tabelę trzeba nadać pozycję względną, zachowując jednocześnie pozycję absolutną elementu zewnętrznego. Jednak z powodu specyfiki działania przeglądarki Internet Explorer elementowi wewnętrznemu można nadać pozycję względną dopiero po utworzeniu tabeli HTML. W przeciwnym razie przewijanie tabeli będzie możliwe, jednak aplikacja nie przytnie jej przy krawędzi głównego kontenera.

Także w tym przypadku jest to najprostsze możliwe zastosowanie suwaków, ale działa. Aby upewnić się, że aplikacja pozostanie dostępna, stronę można zaimplementować w sposób opisany w rozdziale 7., czyli użyć kodu uruchamianego po stronie serwera do pobierania danych i tworzenia tabeli przy użyciu pierwszej grupy wierszy, udostępniając przy tym odnośniki do następnych grup. To działanie powinna zastąpić aplikacja ajaksowa, jeśli użytkownik otworzy stronę za pomocą przeglądarki z włączoną obsługą skryptów. Obie aplikacje mogą używać tej samej usługi sieciowej do pobierania wierszy.

Suwak ma wiele zmiennych elementów, włączając w to rozbudowaną obsługę przeciągania, dlatego ułatwiłam sobie pracę i użyłam biblioteki zewnętrznej — YUI.

Przedstawione tu zastosowanie stronicowania rzadko spotyka się jako efekt Ajaksa. Częściej używane są menu dynamiczne, opisane w następnym podpunkcie.

Menu zgodne z WWW

Menu mogą pojawiać się z boku, opadać z góry, swobodnie pływać po stronie, rozwijać się, zwijać, a czasem pozostają całkowicie niezmienne. Menu to obszar, który wymaga zachowania najwyższej ostrożności, ponieważ w przypadku udostępnienia menu opartego wyłącznie na języku JavaScript użytkownicy bez obsługi skryptów nie będą mieli żadnej możliwości poruszania się po witrynach i aplikacjach sieciowych.

Menu mogą składać się z przycisków i elementów `div`, jednak także w tym przypadku, jeśli programista nie użyje odnośników hipertekstowych, pozbawi możliwości nawigacji osoby, które nie używają myszy.

Można utworzyć skalowalne, hierarchiczne i dynamiczne menu przy zachowaniu dostępnej nawigacji, nie wspominając o poprawności semantycznej. W tym celu należy użyć menu spopularyzowanego przez autorów witryny A List Apart (*http://alistapart.com/articles/dropdowns*), znanego jako menu Suckerfish. Jest ono oparte na liście nieuporządkowanej, pseudoatrybucie `:hover` stylów CSS oraz małej ilości kodu JavaScript, który jest potrzebny, aby rozwiązanie działało w przeglądarkach Internet Explorer 6.x, ponieważ obsługują one pseudoatrybut `:hover` wyłącznie dla znaczników `anchor` (`a`).

Wspomniane menu rozwinięto następnie jako Son of Suckerfish. Nowa wersja umożliwia stosowanie więcej niż dwóch poziomów menu oraz rozwiązuje pewne problemy specyficzne dla różnych przeglądarek (*http://www.htmldog.com/articles/suckerfish/dropdowns*). W przypadku wielopoziomowych menu korzystanie z tego rozwiązania jest nieco skomplikowane i trzeba poradzić sobie z kaskadowym efektem stylów CSS. W oryginalnym menu Suckerfish przy zastosowaniu więcej niż jednego poziomu otwarcie pierwszego poziomu powodowało wyświetlenie wszystkich dodatkowych poziomów. Aby temu zapobiec, trzeba było dodać style CSS ukrywające dodatkowe poziomy po wyświetleniu poziomu pierwszego. Nowsza wersja tego menu to (Son-of-)Suckerfish z obsługą przeglądarki Internet Explorer 7: *http://de.siteof.de/extended-menu-faq-suckerfish-ie7.html*. Technologia posuwa się naprzód, a Suckerfish wciąż ma się dobrze.

To menu jest dostępne, działa bez obsługi skryptów i umożliwia utworzenie hierarchicznego systemu nawigacyjnego na złożonych witrynach.

Menu Suckerfish jest tylko częściowo zgodne z wymogami dostępności, ponieważ użytkownicy korzystający z klawiatury muszą wielokrotnie wciskać klawisz *Tab*.

Niestety, większość systemów do obsługi menu dostępnych w bibliotekach Ajaksa wymaga skryptów. System nawigacyjny to jeden z obszarów, których nie wolno naruszyć przy dodawaniu efektów języka DHTML czy Ajaksa do witryny. Dlatego nie zalecam stosowania żadnych menu DHTML, chyba że mają działać w aplikacji całkowicie zależnej od skryptów. W takiej sytuacji włączenie lub wyłączenie obsługi skryptów to kwestia hipotetyczna.

Przenośne kontenery

Materiały przedstawione w tym punkcie zapożyczyłam z napisanej przeze mnie książki dotyczącej języka JavaScript, *Learning JavaScript*[1] (O'Reilly). Nie przepadam za techniką przeciągania, ponieważ większość jej zastosowań jest niezgodna z oczekiwaniami użytkowników. W trakcie zakupów w sklepie internetowym użytkownik nie oczekuje, że może przeciągać produkty do koszyka. Spodziewa się, że zakupy znajdą się w koszyku po kliknięciu przycisku. W przypadku przenoszenia elementów na stronie w celu dostosowania środowiska pracy może się zdarzyć, że użytkownicy przeciągną pola na inne, początkowo niewidoczne elementy.

Mimo to warto zobaczyć, jak działa przeciąganie, ponieważ czasem zastosowanie tej techniki jest usprawiedliwione. Jednym z takich przypadków jest witryna Google Maps, w której można przeciągnąć mapę w kontenerze i wyświetlić wcześniej niewidoczny obszar. Ta możliwość przeciągania „w obrębie" kontenera, aby zobaczyć zakryte fragmenty, to dobra alternatywa dla obsługi pasków przewijania, nadmiarowych fragmentów i podobnych zagadnień.

Listing 8.3 obejmuje zawartość strony internetowej wraz z arkuszem stylów, zagnieżdżonym w celu uproszczenia przykładu. Warto zwrócić uwagę, że strona zawiera jedynie element `img` umieszczony w elemencie `div`. Rysunek używany w tym przykładzie jest dużo większy niż widoczny fragment, a nadmiarowa część jest ukryta.

Listing 8.3. Okno z przeciąganym widokiem

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pl">
<head>
<title>Efekt podobny do Google Maps</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
#div1
{
   border: 5px solid #000;
   height: 200px;
   left: 100px;
   overflow: hidden;
   position: absolute;
   top: 100px;
   width: 400px;
}
img
{
   border: 1px solid #000;
}
</style>
<script type="text/javascript" src="addingajax.js">
</script>
<script type="text/javascript" src="draggable.js">
</script>
</head>
<body>
<div id="div1" >
<img id="img1" src="robin1.jpg" alt="duży rysunek ciekawskiego drozda" />
</div>
</body>
</html>
```

[1] Wydanie polskie: *JavaScript. Wprowadzenie*, Helion, 2007 — *przyp. tłum.*

Kod do obsługi tego efektu przedstawia listing 8.4. Choć ten kod obsługuje element znajdujący się w kontenerze, można go użyć dla dowolnego elementu. Wystarczy włączyć dla danego obiektu obsługę przeciągania, używając funkcji `makeDraggable` i przekazując do niej identyfikator tego obiektu.

Listing 8.4. Kod JavaScript do obsługi przeciągania w kontenerze

```
// Zmienne globalne
var dragObject;
var mouseOffset;

// Przechwytywanie zdarzeń związanych z myszą
aaManageEvent(document,'mouseup',mouseUp);
aaManageEvent(document,'mousemove',mouseMove);
aaManageEvent(window,'load', function() {
   makeDraggable('img1');
});

// Tworzenie punktu ze współrzędnymi myszy
function mousePoint(x,y) {
   this.x = x;
   this.y = y;
}

// Wykrywanie pozycji myszy
function mousePosition(evnt){
  var x = parseInt(evnt.clientX);
  var y = parseInt(evnt.clientY);
  return new mousePoint(x,y);
}

// Pobieranie przesunięcia elementu na stronie
function getMouseOffset(target, evnt){
   evnt = evnt ? evnt : window.event;
   var mousePos  = mousePosition(evnt);
   var x = mousePos.x - target.offsetLeft;
   var y = mousePos.y - target.offsetTop;
   return new mousePoint(x,y);
}

// Wyłącza przeciąganie
function mouseUp(evnt){
   dragObject = null;
}
// Przechwytuje ruch myszy (tylko przy przeciąganiu)
function mouseMove(evnt){
   if (!dragObject) return;
   evnt = evnt ? evnt : window.event;
   var mousePos = mousePosition(evnt);

   // Jeśli element umożliwia przeciąganie, należy określić nową pozycję bezwzględną
   if(dragObject){
      dragObject.style.position = 'absolute';

      dragObject.style.top      = mousePos.y - mouseOffset.y + "px";
      dragObject.style.left     = mousePos.x - mouseOffset.x + "px";
      return false;
    }
}

// Umożliwia przeciąganie obiektu
function makeDraggable(item){
```

```
    if (item) {
       item = aaElem(item);
       item.onmousedown = function(evnt) {
                          dragObject  = this;
                          mouseOffset = getMouseOffset(this, evnt);
                          return false; };
    }
}
```

Programiści języka JavaScript prawdopodobnie zetknęli się już z przeciąganiem, dlatego nie będę tu opisywać szczegółów działania tej techniki. W witrynie Web Reference znajduje się dobry elektroniczny samouczek na ten temat (*http://www.webreference.com/programming/javascript/mk/column2.html*).

Aplikację udostępniającą przeciąganie w kontenerze można połączyć z utworzonym wcześniej efektem stronicowania i umożliwić przeciąganie tabeli w górę, co przypomina wyjmowanie kartki papieru z teczki w celu przeczytania tekstu znajdującego się na dole. Przeciąganie jest często stosowane do obsługi sortowania, o czym już wspominałam, jednak do zarządzania tym efektem zalecam wykorzystanie gotowych bibliotek.

W tej książce często pojawiają się inne efekty CSS, włączając w to wyróżnianie, okna wyskakujące czy wprowadzanie elementów na stronę. Dostępne są także efekty, które powodują wybuchanie, wyskakiwanie czy przesuwanie się obiektów. Większość bibliotek Ajaksa udostępnia takie funkcje poprzez rozszerzenie, bibliotekę podrzędną lub wtyczkę o nazwie `effects` lub `fx`. W następnym punkcie przejdziemy ze świata stylów CSS do formatu SVG.

Skalowalna grafika wektorowa

Według fragmentu witryny internetowej W3C dotyczącej formatu SVG jest to „… język do opisu dwuwymiarowej grafiki i aplikacji graficznych w formacie XML" (*http://www.w3.org/Graphics/SVG*). Najnowsza udostępniona wersja tej specyfikacji to 1.1, jednak obecnie trwają prace nad poprawianiem projektu wersji 1.2 i dostępne są wersje Print oraz Mobile tej specyfikacji.

Obecnie format SVG jest obsługiwany przez przeglądarki Firefox i Opera, planowane jest dodanie jego obsługi do przeglądarki Safari (w automatycznie kompilowanych wersjach ta obsługa już się pojawia), a firma Adobe udostępniła przeglądarkę umożliwiającą wyświetlanie obrazów SVG w Internet Explorerze.

Przeglądarkę Adobe SVG Viewer dla Internet Explorera można pobrać z witryny firmy Adobe — *http://www.adobe.com/svg/viewer/install/main.html*. Warto jednak wiedzieć, że od 1 stycznia 2008 roku Adobe ma zaprzestać udzielania pomocy technicznej dla tej przeglądarki.

SVG jest oparty na języku XML i składa się z podstawowego zestawu elementów XML, z których każdy ma atrybuty określające sposób wyświetlania. Są dwa sposoby na dołączenie rysunków w formacie SVG. Pierwszy z nich polega na użyciu znaczników SVG w odrębnym dokumencie i zagnieżdżeniu go na stronie. Druga technika to zastosowanie wewnątrzwierszowego kodu SVG.

Zagnieżdżanie kodu SVG

Aby zastosować zagnieżdżanie, należy utworzyć odrębny plik SVG i użyć elementu `embed`, `object` lub `iframe` w celu dołączenia tego pliku do strony internetowej.

Zalecane jest używanie nowszego, standardowego elementu `object`. Przykład użycia tego elementu z plikiem SVG wygląda tak:

```
<object type="image/svg+xml" data="./some.svg" width="200" height="150" ></object>
```

Jeśli programista chce udostępnić alternatywne rozwiązanie dla przeglądarek bez obsługi SVG, powinien umieścić w obrębie znaczników elementu `object` odpowiedni kod HTML:

```
<object type="image/svg+xml" data="./some.svg" width="200" height="150" >
<img src="some.jpg" alt="alternatywny rysunek" />
</object>
```

Niestety, jedyny element, jaki obsługują starsze przeglądarki (mam tu na myśli Netscape 2 i 3), to `embed`. Ten element obsługuje też wtyczka SVG Viewer firmy Adobe. Jeśli programista chce, aby można było wyświetlać obrazy SVG za pomocą tej wtyczki, powinien użyć właśnie elementu `embed`:

```
<embed src="some.svg" width="200" height="150" />
```

Jedyny problem z elementem `embed` polega na tym, że nie wchodzi on w skład specyfikacji HTML, dlatego zawierający go dokument nie przejdzie walidacji jako HTML lub XHTML. Ponadto Internet Explorer nie obsługuje tego elementu i wymaga użycia nowszego elementu `object`.

Na wiki dotyczącej formatu SVG (*http://wiki.svg.org/SVG_and_HTML*) opisana jest technika umożliwiająca rozwiązanie problemu z walidacją. Należy dodać definicje danych, aby zdefiniować zagnieżdżony element i atrybuty na stronie internetowej.

Trzecie podejście umożliwiające rozwiązanie omawianych problemów polega na użyciu ramek `iframe`. Ten element jest obsługiwany we wszystkich docelowych przeglądarkach, a ponadto ma znacznik otwierający i zamykający, co pozwala wstawić alternatywną zawartość używaną w przypadku braku obsługi SVG.

Listing 8.5 przedstawia krótką stronę SVG, która zawiera czerwony okrąg z czarnym obramowaniem. Elementy SVG są opisane w dalszej części rozdziału, jednak w tym miejscu warto wspomnieć, że poniższy kod uruchomiony w walidatorze XML (*http://validator.w3.org*) przejdzie walidację jako dokument SVG 1.1. Należy zwrócić uwagę na użycie typu `DTD SVG` oraz przestrzeni nazw domyślnej dla elementów SVG.

Listing 8.5. Prosty plik SVG z jednym okręgiem

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<ellipse id="ellipse1" cx="210" cy="90" rx="200" ry="80"
```

```
style="fill:rgb(255,0,0);
stroke:rgb(0,0,0);stroke-width:2"/>

</svg>
```

Jeśli serwer internetowy zwraca dokumenty XML, można otworzyć ten plik bezpośrednio w przeglądarce obsługującej format SVG, a wyświetli ona okrąg. Z kolei na stronie internetowej przedstawionej na listingu 8.6 użyto wszystkich trzech elementów służących do otwierania plików SVG: `object`, `embed` i `iframe`.

Listing 8.6. Strona wczytująca plik SVG za pomocą trzech technik

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pl">
<head>
<title>Zagnieżdżony plik SVG</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h3>Element embed</h3>
<embed src="test.svg" type="image/svg+xml" width="600" height="180" />
<h3>Element object</h3>
<object type="image/svg+xml" data="test.svg" width="600" height="180">
<p>Brak obsługi SVG</p>
</object>
<h3>Element iframe</h3>
<iframe src="test.svg" width="600" height="180" frameborder="0">
<p>Brak obsługi SVG</p>
</iframe>
</body>
</html>
```

Można przetestować tę stronę w różnych przeglądarkach. W przeglądarce Firefox 2.0 widoczne są trzy czerwone okręgi. Jest to oczekiwane działanie, ponieważ ta wersja Firefoksa (i używanego w nim silnika Gecko) obsługuje większą część specyfikacji SVG 1.1. Trzy okręgi są widoczne także w przypadku otwarcia strony w przeglądarce Opera 9 i jej nowszych wersjach.

Jednak otwarcie strony w przeglądarce Safari spowoduje wyświetlenie błędu informującego o braku obsługi formatu SVG. Pobranie i zainstalowanie programu SVG Viewer firmy Adobe umożliwia oglądanie plików SVG w Safari, a przeglądarka wyświetli wtedy wszystkie trzy okręgi. Trzy okręgi będą widoczne także po otwarciu strony w przeglądarce WebKit, co sygnalizuje, że w Safari w przyszłości pojawi się wbudowana obsługa plików SVG.

Przejdźmy do komputerów PC. Można sprawdzić działanie strony w przeglądarce Internet Explorer 6.x w systemie Windows 2000. Także w tym przypadku plik SVG będzie niedostępny bez wtyczki firmy Adobe, jednak nawet z nią obsługiwane będą tylko elementy `object` i `iframe`. To samo dotyczy przeglądarki Internet Explorer 7 w systemie Windows XP. Ponieważ `iframe` to prawidłowy element XHTML Transitional (choć już nie Strict), warto ograniczyć się do niego przy zagnieżdżaniu plików SVG.

Dostępna jest wyspecjalizowana wtyczka dla przeglądarki Konqueror, KSVG, wchodząca w skład środowiska KDE. Więcej informacji o niej zawiera strona pomocy technicznej w witrynie *http://svg.kde.org*.

Dodawanie skryptów

Można zrobić wiele, umieszczając kod SVG w odrębnym pliku, który jest ostatecznie zagnieżdżany na stronie. Ponieważ XHTML to prawidłowy kod XML, można go użyć wraz z SVG, co oznacza, że możliwe jest umieszczenie kodu JavaScript w elemencie `script` w celu manipulowania obiektami SVG. Obiekty SVG są częścią modelu DOM w takim samym stopniu co każdy inny element na stronie.

Element `script` można umieścić bezpośrednio w elemencie `svg`, jednak warto dodać sekcję `CDATA` wokół skryptu lub ustawić atrybut `src` na zewnętrzny plik z kodem JavaScript. Programista jest wtedy także bardziej zależny od metod modelu DOM, poziom 2, ponieważ nie ma dostępu do wielu skrótowych metod języka XHTML, które prawdopodobnie zna.

W takiej sytuacji naprawdę przydatne są narzędzia takie jak Firebug z przeglądarki Firefox. Używając opcji *Script* i *DOM*, można łatwo sprawdzić każdy obiekt w kodzie i zobaczyć, jakie właściwości i metody są dostępne. Technika ta działa dla dowolnego dokumentu XML, włączając w to pliki SVG.

Listing 8.7 przedstawia dokument SVG podobny do tego przedstawionego wcześniej, jednak tym razem do elementu `svg` dodano blok `script`. Poniższy kod JavaScript zmienia atrybut `cx` (pozycję w poziomie) elementu, a także jedno z ustawień stylów. Po pierwszym kliknięciu obiektu aplikacja przeniesie go w prawo i zmieni jego kolor na niebieski. Ponowne kliknięcie spowoduje powrót do stanu wyjściowego.

Listing 8.7. Dokument SVG ze skryptem

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<ellipse id="ellipse1" cx="210" cy="90" rx="200" ry="80"
style="fill:rgb(255,0,0);
stroke:rgb(0,0,0);stroke-width:2"/>
<script type="text/javascript">
<![CDATA[
document.getElementById('ellipse1').onclick=changeColor;

function changeColor() {
  var svgObj = document.getElementById('ellipse1');
  var attr = svgObj.attributes.getNamedItem('cx').value;
  if (parseInt(attr) == 210) {
     svgObj.setAttribute('cx',400);
     svgObj.style.setProperty('fill','#0000ff',null);
  } else {
    svgObj.setAttribute('cx',210);
    svgObj.style.setProperty('fill','#ff0000',null);
  }
}
]]>
</script>
</svg>
```

Nie jest potrzebna dodatkowa przestrzeń nazw dla elementu `script`, ponieważ jest on częścią pełnej specyfikacji SVG 1.1. Po otwarciu dokumentu SVG we wcześniejszym testowym dokumencie XHTML kliknięcie dowolnego z trzech okręgów spowoduje przekształcenie danego elementu (oczywiście zależy to od używanej przeglądarki).

Informacje dla osób, które nie korzystały dotąd zbyt często z wymiarów stron: wszystkie wartości powiązane z literą *y* dotyczą osi pionowej, *x* jest powiązana z osią poziomą, a *z* — z prostopadłą do dwóch pozostałych. Inaczej mówiąc — *y* to bok, *x* to podstawa, a *z* biegnie prosto w kierunku twarzy programisty.

Przeglądarka Firefox i te z rodziny Mozilla obsługują pełną specyfikację SVG 1.1 (a przynajmniej jej większą część), podobnie jak Opera i WebKit. Oznacza to, że przedstawiony przykład będzie w nich działał zgodnie z oczekiwaniami. Jedna wtyczka firmy Adobe służąca do wyświetlania rysunków SVG obsługuje jedynie specyfikację ograniczoną (*SVG 1.1 tiny*), która obejmuje tylko podzbiór pełnej specyfikacji, bez elementu `script`.

W ostatnim przykładzie dodano element `script` do dokumentu SVG. A czy można umieścić kod SVG w dokumencie XHTML, a następnie użyć elementu `script` do manipulowania grafiką?

Zagnieżdżony kod SVG

Możliwość wczytania rysunku SVG do elementu `object` jest wygodna, jednak czasem programista chce, aby kod SVG był częścią strony. Można wtedy połączyć go z innymi elementami dokumentu. Tworzenie zagnieżdżonego kodu SVG jest zarówno proste, jak i skomplikowane, a jest tak z powodu różnic między przeglądarkami.

Wiki dotycząca SVG zawiera doskonałą stronę na temat zagnieżdżonego kodu SVG (*http://wiki.svg.org/index.php?title=Inline_SVG*), a w tym miejscu postaram się przedstawić jej praktyczne streszczenie. Jeśli strona zostanie utworzona jako dokument XHTML i jest traktowana jako kod XML, w przeglądarkach Firefox i Opera zagnieżdżony kod SVG będzie przetwarzany poprawnie. Jednak Internet Explorer nie interpretuje kodu XML w podobny sposób i jedynie wyświetla składnię, zamiast ją przetwarzać. Lub mówiąc bardziej precyzyjnie — zamiast użyć wtyczki firmy Adobe do przetworzenia kodu.

Jeśli strona internetowa jest traktowana jako dokument HTML, Internet Explorer interpretuje kod SVG jako następną odmianę języka HTML, co oznacza, że wtyczka przetworzy go prawidłowo. Jednak Firefox (i inne przeglądarki oparte na Gecko) używa odrębnego parsera dla języków HTML i XML, dlatego nie przetworzy kodu SVG w poprawny sposób.

Oznacza to, że jeśli programista użyje rozszerzenia *htm* lub *html* albo typu MIME `HTML`, kod SVG zostanie prawidłowo przetworzony w przeglądarce Internet Explorer, jednak już nie w Firefoksie, Mozilli, Camino, Operze, Safari lub WebKit. Z kolei rozszerzenie *xml* lub *xhtml* działa we wszystkich przeglądarkach oprócz Internet Explorera.

W wiki dotyczącej SVG znajduje się rozwiązanie, dzięki któremu można zwracać strony z zagnieżdżonym kodem SVG działające we wszystkich przeglądarkach, a jest to ta sama technika, która umożliwia prawidłowe przetwarzanie stron XHTML (jej opis znajduje się w rozdziale 1.). To obejście problemu polega na utworzeniu pliku *.htaccess* w katalogu witryny i dodaniu do niego poniższych dyrektyw. Zawierają one instrukcje dotyczące tego, jak serwer ma przesyłać strony internetowe. Jeśli agent użytkownika obsługuje format XHTML, strony są zwracane właśnie w nim. W przeciwnym razie serwer zwraca strony jako dokumenty HTML:

```
AddType text/html .xhtml
RewriteEngine on
RewriteBase /
RewriteCond %{HTTP_ACCEPT} application/xhtml\+xml
RewriteCond %{HTTP_ACCEPT} !application/xhtml\+xml\s*;\s*q=0
```

```
RewriteCond %{REQUEST_URI} \.xhtml$
RewriteCond %{THE_REQUEST} HTTP/1\.1
RewriteRule .* - [T=application/xhtml+xml]
```

W ten sposób można użyć zagnieżdżonego kodu SVG, a ten powinien działać w najbardziej popularnych przeglądarkach.

Pozostałe przykłady zastosowania kodu SVG w tym rozdziale opierają się na tym podejściu zagnieżdżonym, co ułatwia analizę rozwiązań. Teraz przyszła pora, aby bliżej przyjrzeć się temu, jakie efekty można utworzyć za pomocą języka SVG.

Krótki przegląd języka SVG

Kompletna analiza języka SVG wymagałaby całej książki, dlatego w tym miejscu jedynie wspominam o pewnych popularnych elementach i operacjach, co pozwoli na rozpoczęcie korzystania z tego formatu. W kilku poprzednich punktach zobaczyłeś, że element `svg` sam działa jako kontener, do którego można dodawać inne obiekty. Można określić jego szerokość i wysokość, numer wersji czy przestrzeń nazw SVG, a w przypadku umieszczenia go w innym elemencie `svg` można też podać współrzędne *x* i *y*, aby wyznaczyć położenie:

```
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="200">
...
</svg>
```

W tym rozdziale opisuję język SVG w dużym skrócie. Osoby, które chcą go poznać bardziej szczegółowo, zachęcam do wizyty w następujących witrynach: witrynie z samouczkiem SVG Basics (*http://www.svgbasics.com*), witrynie społeczności SVG (*http://svg.org*) i wiki dotyczącej SVG (*http://wiki.svg.org*).

Podstawowe figury i atrybuty

W SVG dostępnych jest kilka podstawowych figur: `ellipse` (elipsa), `circle` (okrąg), `rect` (prostokąt), `line` (linia), `polyline` (linia łamana) i `polygon` (wielokąt). Każdemu z tych obiektów można nadać wysokość i szerokość, a także określić jego lokalizację, zaokrąglić narożniki, wypełnić, użyć kreski różnej grubości, ustalić przezroczystość, przycięcie i tak dalej — oczywiście w zależności od figury. Wszystkie elementy mają także atrybut `style`, w którym można podać style CSS i wyspecjalizowane atrybuty SVG.

Listing 8.8 demonstruje wszystkie podstawowe kształty dostępne w SVG. Dla poszczególnych figur użyto często stosowanych dla nich atrybutów. W przykładzie zastosowałam wewnątrzwierszowy kod SVG. Dokument ma rozszerzenie *xhtml*, a jest pobierany z katalogu, w którym wprowadzono zmiany w pliku *.htaccess*. Ponadto dokument jest zaprojektowany tak, aby przeszedł walidację jako typ `XHTML 1.1 plus MathML 2.0 plus SVG 1.1`. Ten typ dokumentu jest podany na początku pliku.

Listing 8.8. Prawidłowy dokument XHTML z elementami SVG

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
"http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg-flat.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Kod SVG zagnieżdżony w XHTML</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

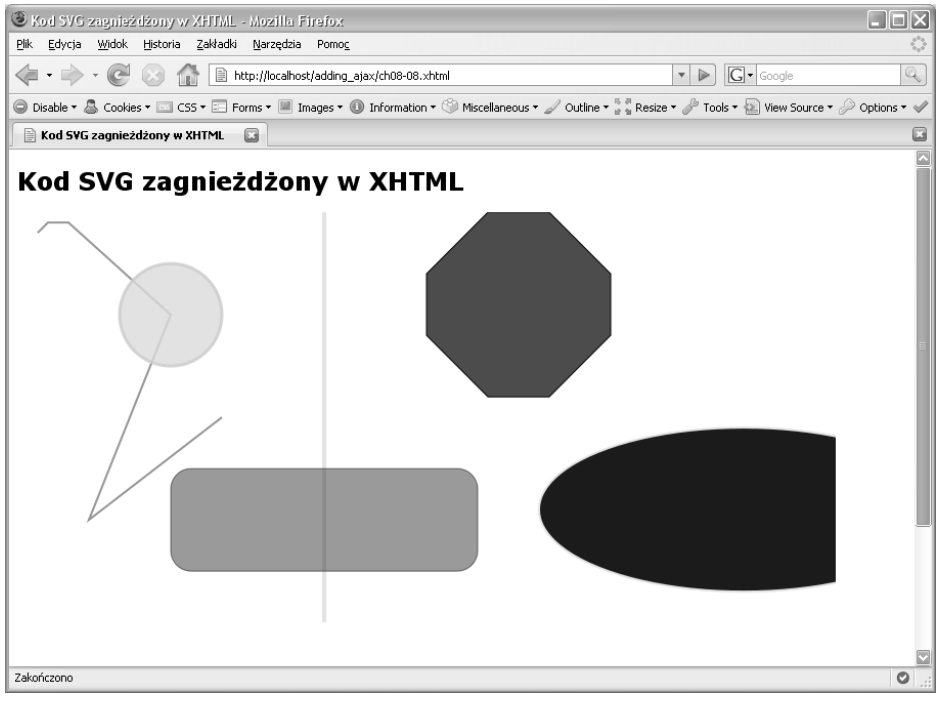
```
  </head>
  <body>
    <h1>Kod SVG zagnieżdżony w XHTML</h1>

    <svg:svg xmlns:svg="http://www.w3.org/2000/svg" width="800" height="600" >

      <svg:polyline points="20,20 30,10 50,10 150,100 70,300 200,200" fill="none"
stroke="#0f0" stroke-width="2" />

      <svg:circle cx="150" cy="100" r="50" fill="#ffcccc" stroke="#ccc" stroke-width="3"
opacity=".8" />

      <svg:line y1="0" x1="300" y2="400" x2="300" stroke="#ff0" stroke-width="4" />

      <svg:polygon points="460,0 520,0 580,60 580,120 520,180 460,180 400,120 400,60"
fill="#f00"
                   style="stroke: #000; stroke-width: 1"/>

      <svg:rect x="150" y="250" rx="20" ry="20" width="300" height="100"
            opacity=".5" fill="purple" stroke="black" stroke-width="1"/>

      <svg:ellipse  cx="710" cy="290" rx="200" ry="80"
              style="fill:rgb(0,0,255);
             stroke:yellow;stroke-width:2"/>

    </svg:svg>

  </body>
</html>
```

Atrybut `DOCTYPE` jest tu przeładowany, jednak umożliwia zastosowanie elementów XHTML, a także SVG i MathML, jeśli programista chce ich użyć, aby uatrakcyjnić wygląd witryny. Zajmijmy się teraz elementami strony. Element `polyline` tworzy linię na podstawie grupy punktów określonych przez pary liczb, które w tym przypadku są rozdzielone przecinkami, choć można także użyć odstępów. Po elemencie `polyline` znajduje się `circle`. Jego środek wyznaczają podane współrzędne `cx` i `cy` (pozycja w poziomie i w pionie), a średnica tego okręgu to `50`. Element `line` przyjmuje punkt początkowy i końcowy, podczas gdy `polygon` funkcjonuje podobnie jak `polyline`, używając podanych punktów, jednak tworzy zamkniętą figurę.

Element `rect` ma tu zaokrąglone narożniki, co określają atrybuty `rx` i `ry`. Ten element oraz `ellipse` (podobnie jak `circle`) są umieszczane w elemencie `svg` na podstawie wartości `x` i `y` oraz `cx` i `cy`. Wszystkie figury mają wartości `fill` i `stroke` (ta ostatnia określa ramkę), a elementy `rectangle` i `circle` są częściowo przezroczyste (atrybut `opacity` ma wartość mniejszą niż 1.0).

Rysunek 8.2 przedstawia wygląd tej strony w przeglądarce Firefox. Dzieło to nie jest niczym specjalnym, jednak pokazuje, jak funkcjonują wszystkie figury względem zawierającego je elementu `svg`. Warto zauważyć, że elipsa, znajdująca się w prawej dolnej części strony, jest przycięta, ponieważ wykracza poza kontener. Należy także zwrócić uwagę na to, że zygzakowaty element `polyline` jest widoczny przez okrąg z uwagi na przezroczystość elementu `circle`. Można także zobaczyć prostą linię za półprzezroczystym prostokątem.

Ten przykład demonstruje różne sposoby dostosowywania elementów, włączając w to zastosowanie stylów i atrybutów SVG. Ponadto warto zauważyć, że w przykładzie użyto nazw kolorów, choć zwykle takie podejście uznawane jest za nieeleganckie. Lepszym rozwiązaniem jest stosowanie ustawień RGB (także zostały użyte w przykładzie) lub wartości w kodzie szesnastkowym (takich jak `#ff0000`; można je spotkać w arkuszach CSS).



Rysunek 8.2. Przykładowy kod SVG w prawidłowym dokumencie XHTML

Elementy def, gradienty, filtry i efekty

Oprócz figur język SVG udostępnia dość bogaty zestaw efektów wizualnych, które można stosować do obiektów tego języka. Wiele z nich jest dołączanych w elementach `def`, które umożliwiają wstępne definiowanie obiektów i efektów używanych w dalszej części dokumentu. Takie elementy są używane zwykle w dokumentach SVG, a nie w postaci zagnieżdżonej w XHTML, można je jednak dołączyć do stron za pomocą opisanych wcześniej technik.

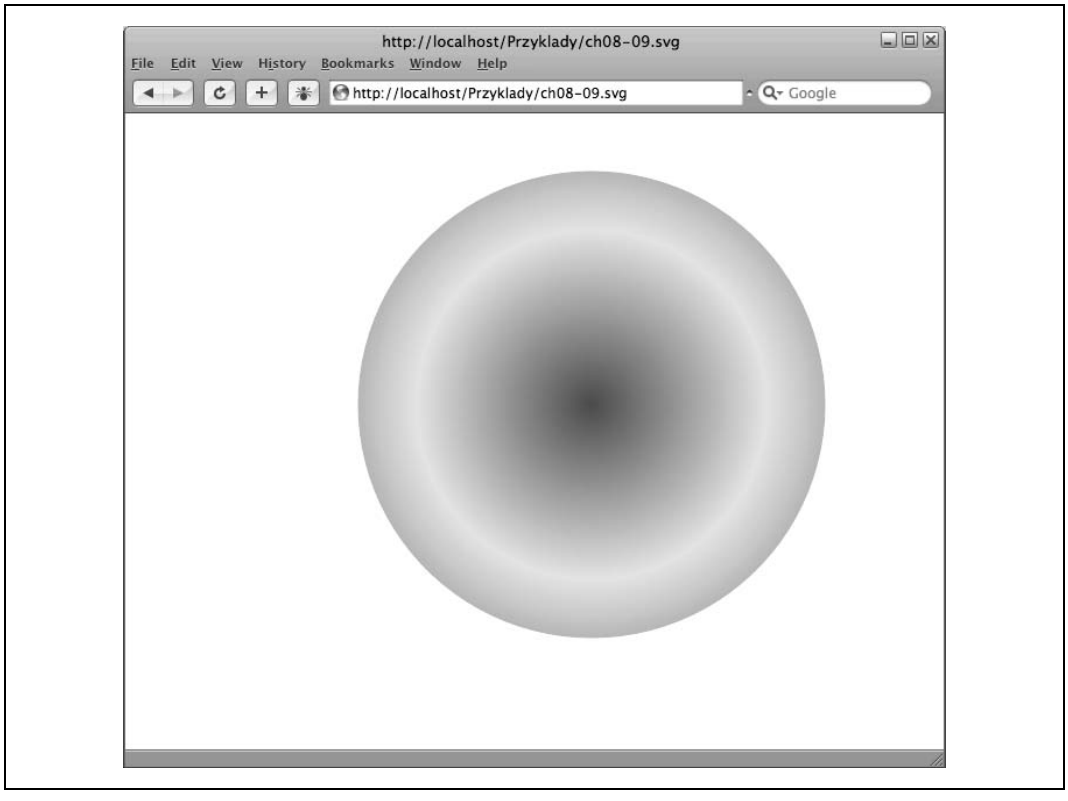
Wstępnie definiowane obiekty i efekty należy umieszczać między otwierającym a zamykającym znacznikiem elementu `def`. Do wyświetlania takich obiektów służy odpowiednio nazwany element `use`. Listing 8.9 przedstawia przykładowy dokument SVG, wyświetlający okrąg zapełniony gradientem, który przechodzi od czerwonego poprzez żółty do ciemnozłotego. Obramowanie tego okręgu ma taki sam odcień złotego.

Listing 8.9. Okrąg wypełniony trójkolorowym gradientem

```
<?xml version="1.0" standalone="no"?>
<svg viewBox="0 0 1100 400"
xmlns="http://www.w3.org/2000/svg" version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
        <circle id = "s1" cx = "200" cy = "200" r = "200" stroke = "#F5B800"
stroke-width = "1"/>
        <radialGradient id = "g1" cx = "50%" cy = "50%" r = "50%">
            <stop stop-color = "#f00" offset = "0%"/>
            <stop stop-color = "#ff0" offset = "75%"/>
```

```
            <stop stop-color = "#f5b800" offset = "100%"/>
        </radialGradient>
    </defs>
    <use x = "200" y = "50" xlink:href = "#s1" fill="url(#g1)"/>
</svg>
```

Jak na kilka wierszy kodu XML, efekt robi wrażenie, a przedstawia go rysunek 8.3. Pamiętaj, że ten rysunek został utworzony za pomocą kodu XML, a nie programu graficznego.



Rysunek 8.3. Okrąg wypełniony gradientem, utworzony przy użyciu kodu SVG

Format SVG to coś, czego brakowało w programowaniu aplikacji WWW od samego początku — wbudowany, działający w różnych przeglądarkach, złożony słownik graficzny, który umożliwia tworzenie wspaniałych efektów bez konieczności korzystania z technologii Flash czy obciążania strony obrazami. Co jeszcze ważniejsze, dzięki możliwości manipulowania elementami za pomocą skryptów można połączyć efekty Ajaksa z grafiką opartą na SVG i tworzyć imponujące prezentacje.

Niektóre opcje języka SVG, na przykład filtry, nie są obsługiwane w pewnych przeglądarkach. W czasie pisania tej książki filtrów nie obsługiwała przeglądarka Firefox 2.x, jednak jej producenci planują dodanie tej funkcji. Firefox umożliwia używanie ścieżek przycinania i masek, jednak te zagadnienia pozostawiam do samodzielnej eksploracji.

Ponieważ określanie układu elementów SVG i próba wyobrażania ich sobie to dość ryzykowane rozwiązanie, dostępne są edytory, które umożliwiają manipulowanie obiektami i generowanie kodu SVG. Jednym z takich programów jest Inkscape (*http://www.inkscape.org*), dostępny

w wersjach dla systemów: Mac OS X, Windows i Linux. Istnieją też inne narzędzia tego typu, a lista wybranych znajduje się na stronie SVGI (*http://www.svgi.org*).

Następny punkt opisuje łączenie SVG i Ajaksa.

Mikser — SVG i Ajax

Obsługa języka SVG w przeglądarkach jest wciąż bardzo młoda, jednak producenci są zaangażowani w jej rozwój, a wraz z pojawianiem się nowych wersji przeglądarek następuje coraz ściślejsza integracja między elementami języka SVG i zwykłymi elementami stron internetowych. W ten sposób chcę powiedzieć, że przykłady łączące SVG i Ajaksa działają głównie w przeglądarkach Firefox i Opera, zwykle w WebKit, a dość rzadko w Internet Explorerze. W tej książce unikam przedstawiania tak ograniczonych rozwiązań, jednak czasem warto pozwolić sobie na nieco zabawy. Na potrzeby aplikacji produkcyjnych należy stosować zagnieżdżony kod JavaScript i skrypty, a jeśli na stronie znajdują się ramki `iframe`, można ich użyć do komunikacji między oboma środowiskami. Warto też ograniczyć stosowanie SVG do elementów, które nie są niezbędne.

Kod na listingu 8.10 jest oparty na przykładzie stronicowania przedstawionym na listingu 8.1 i generuje „kółka" określające popularność. Pod stronicowaną tabelą zagnieżdżony jest mały okrąg utworzony za pomocą SVG. Umieszczenie kursora myszy nad tabelą powoduje wyróżnienie odpowiedniego wiersza przez ustawienie jego tła na kolor czerwony. Po przesunięciu kursora nad inny wiersz kolor poprzedniego zmienia się z powrotem na szary. Aplikacja sprawdza liczbę komentarzy do każdego wpisu. Wartość ta jest zapisana w ostatniej komórce poszczególnych wierszy i służy do określenia wielkości elementu `circle` w kodzie SVG. Brak komentarzy oznacza brak okręgu, a jego wielkość rośnie wraz z ich liczbą.

Zmiany w porównaniu z kodem JavaScript z listingu 8.2 byłyby dość drobne, jednak trzeba poradzić sobie z problemem obsługi różnic między przeglądarkami. Zbyt mało przeglądarek obsługuje zagnieżdżony kod SVG, dlatego trzeba go umieścić w odrębnym pliku, a następnie wczytać do obiektu internetowego. Po pobraniu strony, w zależności od tego, czy używana przeglądarka to Internet Explorer, czy nie, obiekt SVG jest umieszczany albo w elemencie `embed`, albo w `object`:

```
aaManageEvent(window,"load", function() {
  var slider = YAHOO.widget.Slider.getVertSlider("sliderbg",
         "sliderthumb", 0, 400);
  slider.setValue(0,true);
  slider.subscribe("change",adjustPage);

  // Wykrycie przeglądarki Internet Explorer i użycie elementu embed
  if (document.all && !window.opera) {
    var embed = document.createElement('embed');
    embed.src = 'bubble.svg';
    embed.type = 'image/svg+xml';
    embed.width='100%';
    embed.height = '300';
    aaElem('obj').appendChild(embed);

  } else {
    var obj= document.createElement('object');
    obj.type = 'image/svg+xml';
    obj.id = 'svgDoc';
    obj.data = 'bubble.svg';
```

```
    obj.width = '100%';
    obj.height = '300';
    aaElem('obj').appendChild(obj);
  }

  createTable();
  getRows(0);

  aaElem('inner').style.position="relative";
});
```

Po wyświetleniu wierszy aplikacja przypisuje do ich zdarzeń `mouseover` i `mouseout` funkcje obsługi:

```
aaManageEvent(row,'mouseover',showBubble);
aaManageEvent(row,'mouseout',restoreRow);
```

Kiedy uruchomione zostanie zdarzenie `mouseover`, aplikacja wywoła funkcję `showBubble`, a kolor danego wiersza tabeli zmieni się na czerwony. Ponadto aplikacja sprawdza wartość kolumny z liczbą komentarzy i używa jej do zmiany rozmiaru okręgu SVG:

```
function showBubble(evnt) {
   var row = this;
   var val = parseInt(row.lastChild.firstChild.nodeValue);
   val = val * 5;
   row.style.backgroundColor = "#ff0000";

   // Pobieranie w przeglądarce Internet Explorer
   if (document.all && !window.opera) {
      var svgDocument = document.embeds[0].getSVGDocument();
      var svgObject = svgDocument.getElementById('circle1');

   // Pobieranie w pozostałych przeglądarkach
   } else {
      var svgDocument = document.getElementById("svgDoc").contentDocument;
      var svgObject = svgDocument.getElementById('circle1');
   }

   // Ustawianie we wszystkich przeglądarkach
   svgObject.setAttribute("r",val);
}
```

Uruchomienie zdarzenia `mouseout` powoduje ponowną zmianę koloru wiersza tabeli na szary:

```
function restoreRow(evnt) {
   var row = this;
   row.style.backgroundColor="#ccc";
}
```

Listing 8.10 przedstawia stronę internetową z nowym elementem SVG. Także w tym przypadku dodanie kodu SVG do dokumentu XHTML jest niezwykle proste — jeśli tylko dana przeglądarka obsługuje ten format.

Listing 8.10. Przykład stronicowania zintegrowanego z SVG

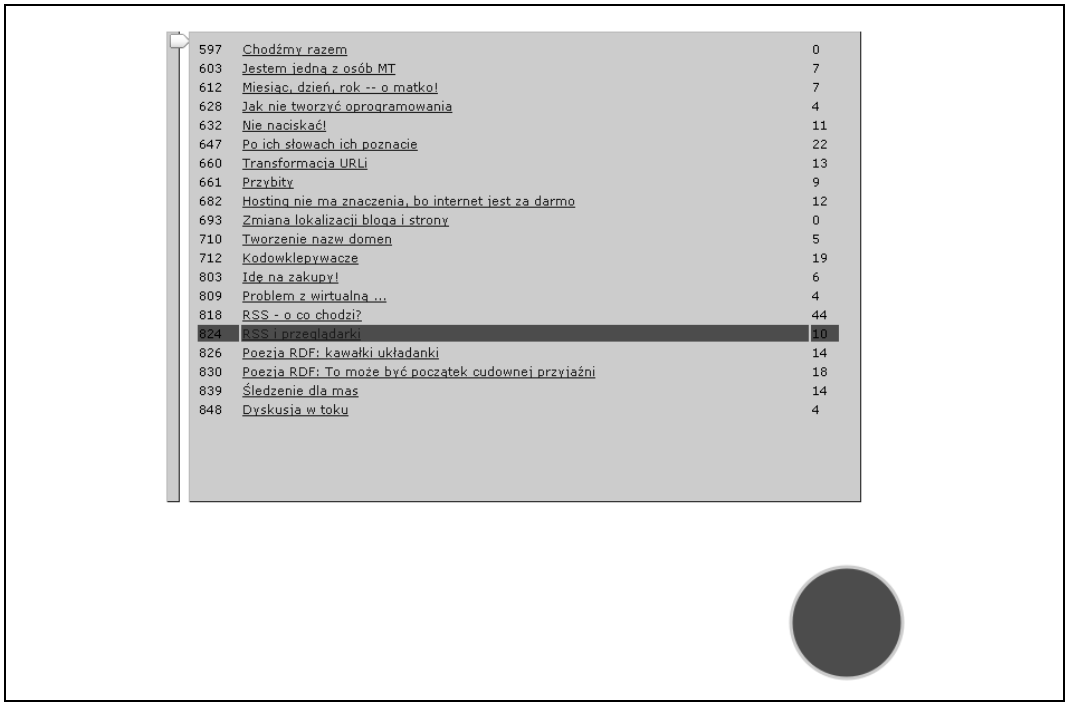
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
"http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg-flat.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pl">
<head>
<title>Bąbelek</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
<link rel="stylesheet" type="text/css" href="pagination.css" />
  <!-- Plik źródłowy z przestrzenią nazw -->
  <script src = "yui/build/yahoo/yahoo.js" ></script>

  <!-- Pliki źródłowe niezbędne ze względu na zależności -->
  <script type="text/javascript" src = "yui/build/dom/dom.js" ></script>
  <script type="text/javascript" src = "yui/build/event/event.js" ></script>
  <script type="text/javascript" src = "yui/build/dragdrop/dragdrop.js" ></script>

  <!-- Plik źródłowy z suwakiem -->
  <script type="text/javascript" src = "yui/build/slider/slider.js" ></script>

  <!-- Pliki specyficzne dla aplikacji -->
  <script type="text/javascript" src="addingajax.js">
  </script>
  <script type="text/javascript" src="paginate2.js">
  </script>

</head>
<body>
<div id="container">
<div id="sliderbg">
  <div id="sliderthumb"><img src=" verticalSlider.gif" alt="uchwyt suwaka"
/></div>
</div>
<div id="main">
<div id="inner">
</div>
</div>
</div>
<div id="obj">
</div>
</body>
</html>
```

Rysunek 8.4 przedstawia tę stronę wyświetloną w Operze. Jest to dość prosty przykład, jednak pokazuje, jak łatwa może być integracja obu omawianych technologii.

Niektóre biblioteki Ajaksa zapewniają obsługę języka SVG. Dojo udostępnia ograniczoną obsługę poprzez bibliotekę dojo.gfx, jednak nie jest ona jeszcze dostępna w wersji produkcyjnej. Działa w różnych przeglądarkach i obejmuje obsługę dla systemu Windows, prawdopodobnie poprzez VML. MochiKit zawiera pakiet do tworzenia wykresów i grafów, PlotKit SVG, który używa obiektu `canvas` języka HTML (opisanego w następnym punkcie) oraz języka SVG.

Google Maps wykorzystuje SVG do tworzenia linii, jednak cofa się do VML w przypadku przeglądarki Internet Explorer. Ponadto Google udostępnia widget SVG w pakiecie GWT (ang. *Google Web Toolkit*). Najważniejszy czynnik ograniczający zastosowania SVG to brak obsługi tego języka w przeglądarce Internet Explorer. Można jednak mieć nadzieję, że niemal powszechna obsługa języka SVG we wszystkich innych przeglądarkach skłoni Microsoft do zapewnienia jej także w Internet Explorerze. Na razie obok wtyczki firmy Adobe dostępne są biblioteki, które przekształcają kod SVG na VML, aby można było użyć go w przeglądarce Internet Explorer. Te biblioteki to między innymi SVG-VML-3D (*www.lutanho.net/svgvml3d/index.html*), JsVectorGraphics (*www.walterzorn.com/jsgraphics/jsgraphics_e.htm*) i RichDraw (*http://starkravingfinkle.org/blog/2006/04/richdraw-simple-vmlsvg-editor*). Dodatkowe informacje na ten temat można uzyskać, wpisując w wyszukiwarce wyrażenie „SVG VML JavaScript Libraries”.



Rysunek 8.4. Integracja Ajaksa z SVG

Obiekt Canvas w HTML5

Apple zaprojektował element `canvas` nie tylko na potrzeby przeglądarki Safari, ale także do użytku w widgetach Dashboard w systemie Mac OS X. Element ten został podchwycony przez grupę WhatWG i został włączony do przyszłego standardu HTML 5.0 — następnej wersji języka HTML. Ten element jest obsługiwany przez przeglądarki oparte na Gecko, takie jak Firefox, przez Safari (oczywiście) i Operę. Internet Explorer nie obsługuje go, jednak trwają prace w celu utworzenia obiektów działających w różnych przeglądarkach (obecnie taki projekt prowadzi Google).

W odróżnieniu od SVG, który jest formalnym dialektem języka XML, element `canvas` to tylko zwykły element. W większości przypadków dostęp do elementu `canvas` odbywa się poprzez model DOM. Element ten służy zwykle do tworzenia pewnych zaawansowanych efektów. Inaczej niż w przypadku SVG, efekty te wymagają stosowania kodu JavaScript.

Projekt Mozilla udostępnia dobry samouczek dotyczący korzystania z obiektu `canvas` (*http://developer.mozilla.org/en/docs/Canvas_tutorial*). Także firma Apple ma podobną stronę (*http://developer.apple.com/documentation/AppleApplications/Conceptual/SafariJSProgTopics/Tasks/Canvas.html*).

Obiekt `canvas` jest dość łatwy w użyciu. Najpierw wystarczy dodać element `canvas` do strony internetowej:

```
<canvas id="graph" width="100%" height="300"></canvas>
```

Aby utworzyć grafikę, należy pobrać kontekst obiektu `canvas`, używając specyficznej dla tego obiektu metody modelu DOM, `getContext`. Najpierw jednak trzeba sprawdzić, czy dana przeglądarka obsługuje ten obiekt:

```
var canvas = aaElem('graph');
if (canvas.getContext) {
   var ctx = canvas.getContext('2d');
   ...
}
```

Po utworzeniu obiektu `canvas` można wykonać przy jego użyciu wiele operacji graficznych: tworzyć figury, rysować ścieżki, tworzyć łuki, manipulować obrazami (umieszczonymi w obiekcie `canvas` na stronie XHTML), tworzyć wykresy funkcji kwadratowych i tak dalej. Po utworzeniu obiektów można, podobnie jak w języku SVG, dodać do nich efekty, takie jak: kolory, wypełnienie, przezroczystość, a nawet utworzyć wzorce, takie jak symbole. Można też zastosować gradienty oraz ostre złączenia między liniami, co jest dość wyjątkowym efektem.

Podobnie jak w SVG, można zarządzać efektami obiektu `canvas` za pomocą Ajaksa. Wracając do przykładu ze stronicowaniem, można zastąpić element `svg` elementem `canvas`:

```
<canvas id="graph" width="1000" height="150"></canvas>
```

W nowej wersji kodu aplikacja pobiera liczbę komentarzy w momencie dodawania wierszy i używa tej wartości do wygenerowania odzwierciedlającej ją linii na dole strony. Służą do tego metody `moveTo` i `lineTo`. Wraz z umieszczaniem danych na stronie linia staje się coraz dłuższa, aż w końcu zajmuje cały dół strony. Listing 8.11 przedstawia kod JavaScript tego rozwiązania.

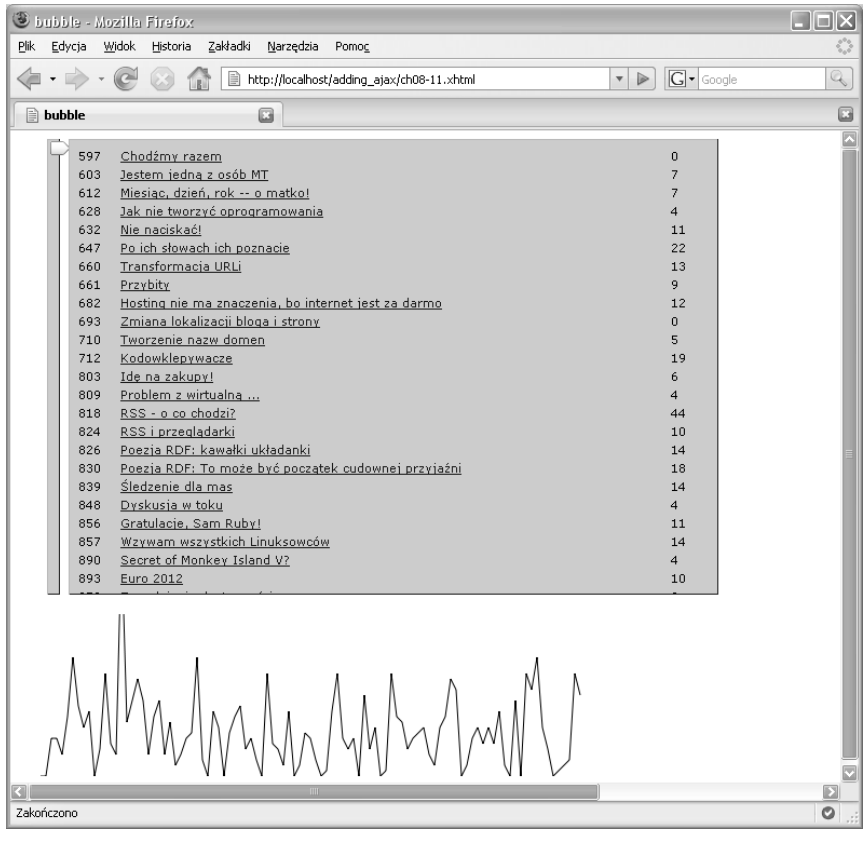
Listing 8.11. Generowanie linii reprezentującej liczbę komentarzy przy użyciu obiektu canvas

```
var startx = 20;
var starty = 150;

var cache = {
   offset : 80,
   fetch : 40
};

aaManageEvent(window,"load", function() {
  var slider = YAHOO.widget.Slider.getVertSlider("sliderbg",
         "sliderthumb", 0, 400);
  slider.setValue(0,true);
  slider.subscribe("change",adjustPage);

  createTable();
  getRows(0);

  aaElem('inner').style.position='relative';
});

// Dostosowuje kontener tabeli na podstawie przesunięcia;
// określa, czy trzeba dodać elementy do pamięci
// podręcznej dla bazy danych
function adjustPage(offset) {

   var inner = aaElem('inner');
   var newTop = -offset;
   inner.style.top = newTop + "px";
```

```
    // Jeśli faktyczne przesunięcie jest większe niż w pamięci
    // podręcznej, należy pobrać wiersze i zaktualizować tę pamięć
    if (offset > cache.offset) {
       getRows(cache.fetch);
       cache.offset+=80;
       cache.fetch+=40;
    }
}
// Pobiera brakującą grupę wierszy
function getRows(start) {
    var url = 'getposts.php?start=' + start;
    var script = document.createElement('script');
    script.type = 'text/javascript';
    script.src = url + '&limit=40&callback=printRows';
    document.getElementsByTagName('head')[0].appendChild(script);
}
// Tworzy i dołącza komórkę tabeli
function createTableCell(value,tr) {

    // Tworzy element <td>
    var cell = document.createElement("td");

    // Tworzy węzeł tekstowy
    var text = document.createTextNode(value);

    // Dołącza utworzony węzeł tekstowy do komórki <td>
    cell.appendChild(text);

    // Dołącza komórkę <td> do wiersza <tr>
    tr.appendChild(cell);

    return tr;
}

// Tworzy pustą tabelę i dodaje ją do strony;
// ustawia właściwość top kontenera
function createTable() {
    var inner = aaElem('inner');

    var table = document.createElement('table');
    table.id = 'dataTable';
    var tableBody = document.createElement("tbody");
    tableBody.id = "dataTableBody";
    table.appendChild(tableBody);
    inner.appendChild(table);
    inner.style.top = "0px";
}
// Dołącza nowo pobrane wiersze jako wiersze tabeli
function printRows(rowsObj) {

    // Usuwanie dawnej zawartości
    var tableBody = aaElem('dataTableBody');

    // Wiersze tabeli
    for(var i = 0; i < rowsObj.length; i++) {
         // Tworzy element <tr>
         var row = document.createElement("tr");
         row = createTableCell(rowsObj[i].id, row);

         // Tworzy element anchor oraz tytuł
         var cell = document.createElement("td");
         cell.innerHTML = "<a href='" + rowsObj[i].guid + "'>" + rowsObj[i].title +
                          "</a>";
```

```
            row.appendChild(cell);

            row = createTableCell(rowsObj[i].comments,row);

           tableBody.appendChild(row);
      }
      var canvas = aaElem('graph');
      if (canvas.getContext) {
         var ctx = canvas.getContext('2d');
         for (var j = 0; j < rowsObj.length; j++) {
            ctx.moveTo(startx,starty);
            startx+=5;
            starty = 150 - (parseInt(rowsObj[j].comments) * 5);
            ctx.lineTo(startx, starty);
         }
         ctx.stroke();
      }
   }
```

Zmodyfikowane fragmenty kodu są wyróżnione pogrubieniem, co demonstruje, jak łatwo jest użyć obiektu `canvas`. Rysunek 8.5 przedstawia działanie tego przykładu w przeglądarce Safari, która obsługuje obiekt `canvas`, choć na razie nie zapewnia obsługi języka SVG.



Rysunek 8.5. Stosowanie obiektu canvas do śledzenia liczby komentarzy

Przedstawiony materiał to tylko krótkie wprowadzenie. Doskonałym przeglądem wszystkich możliwości obiektu `canvas` jest wspomniany już samouczek w witrynie projektu Mozilla.

Przyszłość grafiki

Grafika to ostatnia bariera dla stron internetowych, jednak na szczęście widać duży postęp, jeśli chodzi o możliwości dostępne programistom. Wszystkie przeglądarki obsługują przynajmniej znaczący podzbiór stylów CSS, a także co najmniej jedną z rozszerzonych bibliotek graficznych lub jeden z takich obiektów. Niestety, nie wszystkie przeglądarki obsługują każdą bibliotekę. Firefox i Opera obsługują najwięcej mechanizmów, choć WebKit sprawia, że można wiązać duże oczekiwania z Safari, a Microsoft czyni kroki w kierunku zbliżenia Internet Explorera do pozostałych przeglądarek.

Dzięki pomysłowości programistów większość funkcji graficznych przedstawionych w tym rozdziale ma obsługę w postaci bibliotek działających w różnych przeglądarkach, a obsługa ta wykracza poza dość proste przykłady zaprezentowane w tym miejscu.

Z używaniem jakiejkolwiek grafiki wiążą się pewne problemy, z których wcale nie najmniej istotnym jest to, że wymaga ona elementów „wizualnych", dlatego dla osób z upośledzeniem wzroku takie efekty nie będą użyteczną nowością w witrynie. Jednak jak ilustrują to przedstawione przykłady, grafika może być zabawnym i interesującym dodatkiem do istniejących aplikacji, zwłaszcza w postaci wizualnych wykresów dla danych dostępnych już jako tekst. Większość zastosowań jest zależna od skryptów, jednak także w tym przypadku brak ich obsługi sprawia, że efekt po prostu nie zostanie wyświetlony, a aplikacja nie straci na funkcjonalności.

Jeśli chodzi o różnice między przeglądarkami, takie jak w ostatnim przykładzie, jeśli przeglądarka nie obsługuje obiektu `canvas`, aplikacja nie wyświetli grafiki. Jednak wciąż dostępna będzie tabela z danymi.

Wszystkie zaawansowane efekty wizualne i obiekty omówione w tym rozdziale, jeśli używa się ich jako ciekawych dodatków, pozwalają dodać stronie wiele uroku stosunkowo małym kosztem.