Po prostu JavaScript i Ajax. Wydanie VI

Autorzy: Tom Negrino, Dori Smith
Tłumaczenie: Wojciech Moch, Łukasz Orzechowski
ISBN: 978-83-246-0839-3
Tytuł oryginału: JavaScript and Ajax for the Web, Sixth Edition: Visual QuickStart Guide (6th Edition)
Format: B5, stron: 524

Bezproblemowe wprowadzenie do języka JavaScript

- Chcesz budować bardziej interaktywne witryny internetowe?
- Chcesz zobaczyć, jak dynamicznie manipulować elementami stron?
- Chcesz dowiedzieć się, jak ulepszyć interfejs użytkownika za pomocą technologii Ajax?

W życiu każdego webmastera nadchodzi moment, w którym zwykły kod HTML oraz arkusze CSS już nie wystarczają i pora poszukać bardziej zaawansowanych narzędzi. Jeśli sięgnąłeś po tę książkę, prawdopodobnie pragniesz też tchnąć w swe witryny więcej życia i dynamiki. Naturalną drogą rozwoju jest nauka języka JavaScript, który umożliwia budowanie bardziej interaktywnych i efektownych stron internetowych. A skoro już zaczniesz poznawać ten język, dlaczego nie wypróbować bazującej na nim technologii Ajax, która pozwala tworzyć jeszcze ciekawsze i bardziej wygodne interfejsy użytkownika.

Dzięki książce „o prostu JavaScript i Ajax. Wydanie VI" błyskawicznie nauczysz się wykorzystywać JavaScript do poprawy jakości własnych witryn internetowych. Poznasz podstawy składni tego języka, sposoby zagnieżdżania skryptów w kodzie HTML, techniki dynamicznego manipulowania rozmaitymi elementami stron internetowych oraz reagowania na zdarzenia zachodzące w przeglądarce. Dowiesz się też, do czego służy technologia Ajax oraz jak wykorzystać jej możliwości do tworzenia efektownych i szybkich interfejsów użytkownika.

- Podstawy języka JavaScript
- Dodawanie rysunków i animacji
- Korzystanie z ramek
- Sterowanie oknami przeglądarki
- Stosowanie formularzy
- Obsługa zdarzeń
- Używanie ciasteczek
- Stosowanie arkuszy CSS
- Manipulowanie modelem DOM
- Korzystanie z technologii Ajax
- Tworzenie skryptozakładek

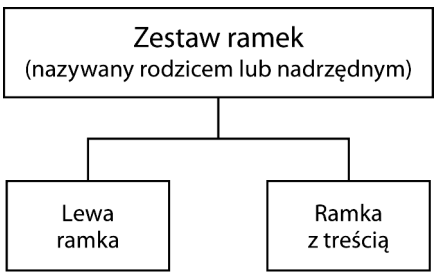
Poznaj praktyczne zastosowania języka JavaScript i twórz lepsze witryny internetowe

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

Spis treści

Ramki, ramki i jeszcze raz ramki

5



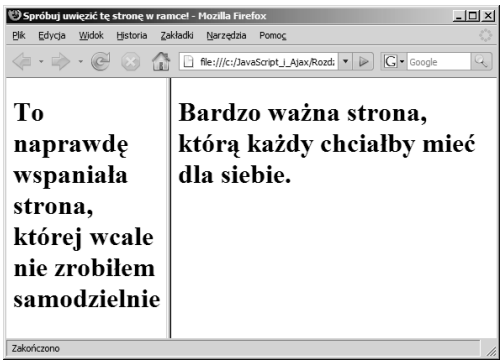
Rysunek 5.1. *Układ zestawu ramek składającego się z dwóch ramek: lewej ramki oraz ramki treści*

Ramki to jeden z najbardziej efektywnych elementów HTML, choć w ostatnich latach dość mocno wypadły z łask twórców stron WWW. W tym rozdziale opisujemy, jak ujarzmić je za pomocą JavaScriptu i jeszcze bardziej rozszerzyć ich możliwości.

Zestaw ramek składa się z co najmniej trzech dokumentów HTML. Pierwszy z nich nazwany *frameset*, czyli właśnie zestaw ramek, zawiera ustawienia wszystkich ramek potomnych. W kodzie JavaScriptu, odwołując się do niego, korzystamy z nazwy `top` lub `parent`. Pozostałe strony, nazywane stronami potomnymi, dostosowują się do układu stworzonego przez zestaw ramek. Stronom tym można przypisywać dowolne nazwy. Rysunek 5.1 przedstawia zestaw ramek z dwoma ramkami potomnymi o nazwie *menu* i *treść*.

Tabela 5.1. *Podstawy HTML — Ramki*

Znacznik	Atrybut	Znaczenie
`frameset`		Określa, że strona składa się z dwóch lub więcej stron umieszczonych w ramkach. Zawiera znaczniki `<frame>`, które definiują poszczególne strony.
	`cols`	Wymiary kolumn poszczególnych ramek (proporcjonalne lub stałe) podawane w pikselach.
`frame`		Opisuje położenie i atrybuty strony w ramce.
	`id`	Stosowany w języku JavaScript do odwoływania się do jednej ze stron w zestawie ramek.
	`name`	Inna metoda pozwalająca językowi JavaScript odwoływać się do jednej ze stron w zestawie ramek.
	`src`	Adres URL strony, która ma się pojawić w tej ramce.
`iframe`		Ramka wewnętrzna, wyświetlana wewnątrz wywołującej ją strony HTML.
	`id`	Stosowany w języku JavaScript do odwoływania się do stron w ramkach.
	`name`	Inna metoda pozwalająca językowi JavaScript odwoływać się do stron w ramkach.
	`src`	Adres URL strony, która ma się pojawić w ramce.
	`width`	Szerokość ramki określana w pikselach lub w procentach.
	`height`	Wysokość ramki określana w pikselach lub w procentach.
	`align`	Wyrównuje ramkę do lewej lub do prawej strony.
	`framebodrer`	Wyświetla obramowanie wokół ramki.



Rysunek 5.2. *Nasza strona jako część strony innej osoby*

Skrypt 5.1. *Oto kod HTML strony, którą chcemy zabezpieczyć*

Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Nie może znaleźć się
  ➥w ramce</title>
  <script language="javascript"
  ➥type="text/javascript"
  ➥src="script01.js"></script>
</head>
<body bgcolor="#FFFFFF">
  <h1>Bardzo ważna strona, którą każdy
  ➥chciałby mieć dla siebie.</h1>
</body>
</html>
```

Skrypt 5.2. *Kod JavaScript wymuszający wyświetlanie naszej strony w osobnym oknie*

Skrypt

```
if (top.location != self.location) {
  top.location.replace(self.location);
}
```

Zapobieganie wyświetleniu strony w ramce

Inne osoby mogą umieścić naszą stronę w ramce będącej częścią ich witryny, stwarzając wrażenie, że stanowi ona część ich serwisu WWW. W JavaScripcie okna są ułożone w strukturze hierarchicznej, na szczycie której znajduje się okno `parent`. Jeśli ktoś próbuje „uprowadzić" naszą stronę, oznacza to, że wyświetla ją w ramce potomnej w stosunku do swojego okna `parent`. Na rysunku 5.2 można zobaczyć, jak wyglądałaby nasza strona jako część czyjejś witryny. Za pomocą skryptu 5.1 i 5.2 można zapobiec takiej sytuacji i zmusić przeglądarkę do tego, aby nasza strona zawsze była wyświetlana w osobnym oknie. Skrypt 5.1 zawiera kod strony HTML, którą chcemy w ten sposób zabezpieczyć (proszę zwrócić uwagę na znacznik `<script>`). Z kolei skrypt 5.2 zawiera kod JavaScript, którym zajmiemy się za chwilę.

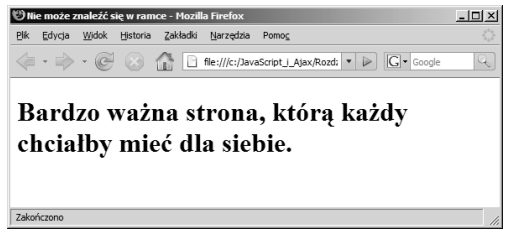
Aby odizolować stronę WWW:

1. `if (top.location != self.location) {`

   Najpierw sprawdzamy, czy aktualna strona (`self`) znajduje się na szczycie hierarchii okien przeglądarki. Jeśli tak, to nie ma potrzeby, aby cokolwiek robić.

2. `top.location.replace(self.location);`

   Jeśli aktualna strona nie znajduje się na szczycie hierarchii, to przypisujemy aktualną stronę do szczytu. Sprawi to, że nasza strona zostanie wyświetlona samodzielnie. Na rysunku 5.3 zobaczyć można stronę, tak jak planowaliśmy — w osobnym oknie.



Rysunek 5.3. *Nasza strona po udanej „ucieczce" z innej strony*

Wskazówka

- Moglibyśmy po prostu przypisać wartość `top.location` do właściwości `self.location`, ale takie postępowanie miałoby pewien niemiły efekt uboczny: użytkownik nie mógłby skorzystać w przeglądarce z przycisku *Wstecz*. Naciśnięcie tego przycisku powodowałoby automatyczny powrót do bieżącej strony. Dzięki zastosowaniu metody `replace()` podmieniamy po prostu aktualną stronę w historii, dzięki czemu przycisk *Wstecz* może działać bez żadnych komplikacji.

Skrypt 5.3. *Strona z zestawem ramek*

Skrypt

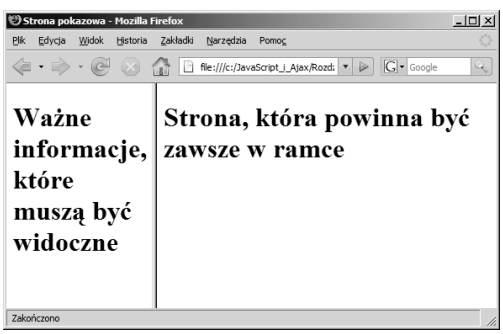
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Frameset//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Strona pokazowa</title>
</head>
<frameset cols="30%,70%">
  <frame src="left2.html" name="left"
  ➥id="left" />
  <frame src="frame2.html" name="content"
  ➥id="content" />
</frameset>
</html>
```

Skrypt 5.4. *Kod JavaScript wymuszający wyświetlenie strony w ramkach*

Skrypt

```
if (top.location == self.location) {
  self.location.replace("frameset2.html");
}
```

Rysunek 5.4. *Samotna strona w oknie przeglądarki*



Rysunek 5.5. *Nasza strona połączona z rodzicem i rodzeństwem*

Umieszczenie strony w ramce

Automatyczne programy katalogujące strony internetowe nie wiedzą, że strony te stanowią część zestawu ramek. Gdy użytkownik odnajdzie jedną z naszych stron za pośrednictwem wyszukiwarki internetowej, to kliknięcie łącza zaprowadzi go do pojedynczej strony, a nie do strony będącej częścią zestawu ramek, tak jak to sobie zaplanowaliśmy. Skrypt 5.3 zawiera kod HTML strony z zestawem ramek. Strona, którą użytkownik otrzymał od przeglądarki, zawiera wywołanie skryptu JavaScript (podobnie jak w skrypcie 5.1). Skrypt 5.4 demonstruje, jak wyświetlić stronę w ramce, nawet jeśli o ramkach nie „wie" Google czy też inna wyszukiwarka internetowa. Na rysunku 5.4 można zobaczyć stronę wyświetloną nieprawidłowo, a na rysunku 5.6 — stronę poprawnie umieszczoną w ramkach.

Aby wymusić wyświetlanie strony w ramce:

1. `if (top.location == self.location) {`

   Sprawdzamy, czy aktualna strona (`self`) znajduje się na szczycie hierarchii. Jeżeli nie, to znaczy, że jest wyświetlana w zestawie ramek.

2. `top.location.replace("frameset2.html");`

   Jeżeli aktualna strona znajduje się na szczycie hierarchii, to zastępujemy ją adresem URL zestawu ramek. Wówczas wyświetlone zostaną ramki, a w nich nasza strona, tak jak zaplanowaliśmy.

Wskazówka

- Sposób ten jest użyteczny tylko wówczas, gdy stosuje się go w niewielkich witrynach internetowych, ponieważ wymaga on, by każda umieszczana w ramkach strona posiadała swój własny plik z zestawem ramek. Rozwiązanie przydatne w przypadku większych witryn przedstawimy w następnym przykładzie.

Umieszczenie strony w ramce — rozwiązanie dla dużych witryn

Jeżeli posiadamy dużą witrynę internetową zawierającą wiele stron, które powinny znajdować się w ramce, to stosowanie opisanego wcześniej sposobu szybko stałoby się bardzo niewygodne. Istnieje inne rozwiązanie, przydatne dla większych witryn. W skrypcie 5.5 przedstawiliśmy definicję ramek używaną w tym rozwiązaniu, a w skrypcie 5.6 — wywoływany przez niego kod JavaScript. Na rysunku 5.6 można zobaczyć samą stronę, natomiast na rysunku 5.7 został przedstawiony oczekiwany wygląd wyświetlanej strony. Nie prezentujemy tu kodu HTML lewej części strony widocznej na rysunku 5.7 (pasek nawigacyjny) ani równie prostej strony, stanowiącej treść całej strony. Samo wywołanie kodu JavaScript wygląda dokładnie tak samo jak w skrypcie 5.5.

Aby wymusić wyświetlanie ramek dla całej witryny internetowej:

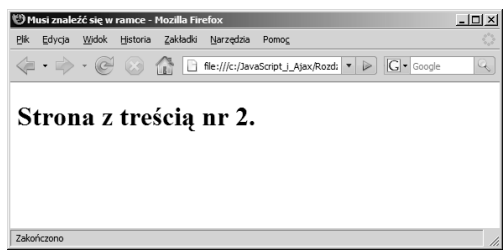
1.
```
var framesetPage = "frameset3.html";
var currPage = justTheFilename
➥(self.location.pathname);
```

   Zaczynamy od utworzenia dwóch zmiennych: `framesetPage` i `currPage`. Pierwsza z nich opisuje stronę układu ramek, którą chcemy narzucić odgórnie w całej witrynie, i dlatego otrzymuje ona wartość `frameset3.html`. Druga zmienna zawiera nazwę strony HTML, która wywołała ten zewnętrzny plik JavaScript. Nazwa ta musi zostać wymieniona, ponieważ język JavaScript nie obejmuje funkcji, która byłaby w stanie zwrócić taką wartość. W tym celu wykorzystamy funkcję `justTheFilename()`, która będzie opisywana w kroku 10.

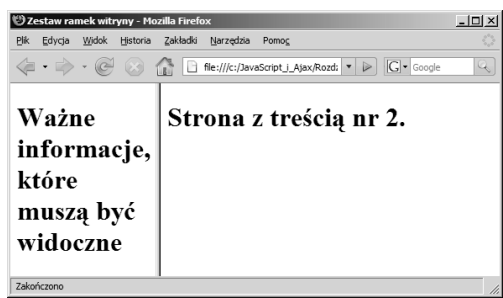
Skrypt 5.5. *Strona z zestawem ramek wywołująca zewnętrzny kod JavaScript*

Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Frameset//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Zestaw ramek witryny</title>
  <script language="javascript"
  ➥type="text/javascript"
  ➥src="script03.js"></script>
</head>
<frameset cols="30%,70%">
  <frame src="left3.html" name="left"
  ➥id="left" />
  <frame src="frame3a.html" name="content"
  ➥id="content" />
</frameset>
</html>
```



Rysunek 5.6. *Nasza strona bez towarzyszących ramek*



Rysunek 5.7. *Planowany przez nas wygląd witryny*

Skrypt 5.6. *Kod JavaScript wymuszający umieszczenie stron witryny w wybranych przez nas ramkach*

Skrypt

```
var framesetPage = "frameset3.html";
var currPage = justTheFilename
➥(self.location.pathname);

if (top.location == self.location &&
➥framesetPage != currPage) {
  self.location.replace(framesetPage + "?" +
  ➥currPage);
}

window.onload = chgFrame;

function chgFrame() {
  if (top.location == self.location &&
  ➥document.location.search) {
    var linkURL = justTheFilename
    ➥(document.location.search);
    var contentWin = document.getElementById
    ➥("content").contentWindow;
    var currURL = justTheFilename
    ➥(contentWin.location.pathname);

    if (currURL != linkURL) {
      contentWin.location.replace(linkURL);
    }
  }
}

function justTheFilename(thisFile) {
  if (thisFile.indexOf("/") > -1) {
    thisFile = thisFile.substring
    ➥(thisFile.lastIndexOf("/")+1);
  }

  if (thisFile.indexOf("?") == 0) {
    thisFile = thisFile.substring(1);
  }

  return thisFile;
}
```

W tym kroku funkcji przekazujemy wartość `self.location.pahtname` (właściwość przechowującą część adresu URL znajdującą się za nazwą domeny). Na przykład jeżeli oglądalibyśmy stronę *http://www.helion.pl/index.html*, to właściwość `self.location.pathname` miałaby wartość `/index.html`. Jeżeli oglądalibyśmy stronę *http://www.helion.pl/ksiazki/index.html*, to właściwość ta miałaby wartość `/ksiazki/index.html`. W obu przypadkach chodzi nam wyłącznie o nazwę pliku *index.html*, wyliczaną i zwracaną przez funkcję `justTheFilename()`.

2.
```
if (top.location == self.location &&
➥framesetPage != currPage) {
  self.location.replace(framesetPage +
  ➥"?" + currPage);
}
```

Teraz wykonywana jest typowa czynność sprawdzająca, czy `top.location` ma taką samą wartość jak `self.location`. Taką kontrolę wykonywaliśmy już wcześniej, ale tym razem uzupełniliśmy ją o dodatkowy warunek sprawdzający: czy znajdujemy się na stronie zestawu ramek. Jeżeli tak, to świetnie, bo nie musimy ponownie niczego ładować. Jeżeli jednak zmienna `currPage` nie jest stroną zestawu ramek, to znaczy, że coś nie działa jak należy. Musimy zatem ponownie załadować stronę zestawu ramek, przekazując jej stronę `currPage`, tak żeby trafiła ona we właściwe miejsce.

3.
```
window.onload = chgFrame;
```

Funkcja obsługi zdarzenia `onload` definiowana jest dopiero tutaj, ponieważ chcemy, żeby funkcja `chgFrame()` wywoływana była przez wszystkie strony korzystające z zewnętrznego skryptu JavaScript.

4.
```
function chgFrame() {
```

Ta funkcja sprawdza, czy (a) znajdujemy się we właściwym zestawie ramek, (b) w adresie URL znalazł się znak zapytania poprzedzający nazwę pliku. Jeżeli tak, to znaczy, że plik ten musi zostać załadowany do ramki treści.

5. `if (top.location == self.location &&`
   `➥document.location.search) {`

   Ponownie wykonywane są typowe czynności sprawdzające, czy `top.location` ma wartość identyczną z `self.location`. Jeżeli tak, to znaczy, że jesteśmy w zestawie ramek. Następnie sprawdzana jest zawartość właściwości `document.location.search`. Jest to kolejne wbudowane w język pole, zawierające część adresu URL zaczynającą się od znaku zapytania (o ile taka istnieje). Jeżeli takiego znaku nie ma, to pole `document.location.search` będzie puste, co spowoduje zakończenie funkcji.

6. `var linkURL = justTheFilename`
   `➥(document.location.search);`

   Oto pierwsza z trzech zmiennych, które musimy przygotować przed załadowaniem nowych treści do ramki. Najpierw zmiennej `linkURL` przypisujemy nazwę pliku, który ma zostać załadowany do ramki. Pomocna jest tutaj funkcja `justTheFilename()`, której przekazujemy wartość pola `document.location.search`.

7. `var contentWin = document.getElementById`
   `➥("content").contentWindow;`

   Zmienna `contentWin` musi przechowywać informacje na temat docelowej ramki. Szukamy zatem elementu o identyfikatorze `content` i pobieramy z niego wartość właściwości `contentWindow`. Opisuje on stronę załadowaną do ramki.

8. `var currURL = justTheFilename`
   `➥(contentWin.location.pathname);`

   Zmiennej `currURL` przypisujemy stronę HTML aktualnie załadowaną do ramki z treścią. Tutaj również wykorzystywana jest funkcja `justTheFilename()`, której tym razem przekazujemy wartość właściwości `contentWin.location.pathname`.

9.
```
if (currURL != linkURL) {
  contentWin.location.replace(linkURL);
}
```

W tym miejscu mamy już wszystkie potrzebne informacje i moglibyśmy ponownie załadować ramkę z treścią, ale dopisując jeszcze jedną instrukcję, możemy wprowadzić bardzo sprytne rozwiązanie. Po co zajmować się ładowaniem tej samej strony, która aktualnie znajduje się w ramce z treścią? Wystarczy, że porównamy ze sobą zawartość zmiennych `currURL` i `linkURL`. Jeżeli są identyczne, to nie musimy nic robić i pozostawiamy stronę bez zmian. Jeżeli jednak wartości tych zmiennych są różne, to wywołujemy znaną nam już metodę `replace()`.

10.
```
function justTheFilename(thisFile) {
  if (thisFile.indexOf("/") > -1) {
    thisFile = thisFile.substring
    ➥(thisFile.lastIndexOf("/")+1);
  }
  if (thisFile.indexOf("?") == 0) {
    thisFile = thisFile.substring(1);
  }
  return thisFile;
}
```

Do omówienia pozostała nam już tylko funkcja `justTheFilename()`. Funkcja ta pobiera ciąg znaków i próbuje wyodrębnić z niego nazwę pliku. Najpierw sprawdzamy, czy w ciągu znaków znajduje się znak ukośnika (/). Jeżeli jest, to szukamy ostatniego ukośnika w ciągu (metoda `lastIndexOf()`) i za nazwę pliku przyjmujemy wszystko to, co znajduje się za nim. Następnie szukamy znaku zapytania. Jeżeli będzie to pierwszy znak nazwy pliku (na pozycji zerowej), to za nazwę pliku uznajemy cały tekst od pozycji pierwszej aż do końca. Tak wyodrębniona nazwa pliku jest zwracana do funkcji wywołującej.

Załadowanie ramki

Często jednej z ramek używa się jako paska nawigacyjnego, za pomocą którego różne strony są ładowane do głównej ramki. Główna ramka staje się elementem docelowym dla łączy znajdujących się w ramce nawigacyjnej. W celu załadowania stron do tej ramki za pomocą kodu HTML stosuje się atrybuty `target` umieszczane w znacznikach `<a>`. Jeżeli jednak chcielibyśmy używać języka XHTML w wersji Strict, to jedyną metodą ładowania stron do ramek jest wykorzystanie języka JavaScript. Po prostu język XHTML Strict nie pozwala na stosowanie atrybutów target, a zatem trzeba w inny sposób podać element docelowy ładowania strony.

W skrypcie 5.9 pokazujemy kod zestawu ramek w języku XHTML Strict, w skrypcie 5.7 dostępny jest kod strony menu nawigacyjnego, a kod JavaScript, wywoływany po kliknięciu dowolnego łącza, został przedstawiony w skrypcie 5.8. Samą stronę można zobaczyć na rysunku 5.8.

Aby wyznaczyć element docelowy ładowanej strony:

1. `window.onload = initLinks;`

   Po załadowaniu strony wywoływana jest funkcja `initLinks()`.

2. ```
   function initLinks() {
     for (var i=0; i<document.links.
     ➥length; i++) {
       document.links[i].target = "content";
     }
   }
   ```

   Ta funkcja przegląda wszystkie łącza na stronie. Dla każdego znalezionego łącza definiowana jest właściwość `target`, której przypisywany jest ciąg znaków `content`.

Wskazówka

- Jeżeli język JavaScript zostanie wyłączony, to użytkownicy klikający dowolne łącze zostaną zaskoczeni. Strona docelowa zostanie załadowana do ramki paska nawigacyjnego, a nie do ramki treści. Niestety, tak działają ramki w języku XHTML Strict.

Skrypt 5.7. *Jeżeli w zestawie ramek używamy języka XHTML Strict, to do zdefiniowania docelowych ramek dla łączy trzeba wykorzystać język JavaScript*

Skrypt

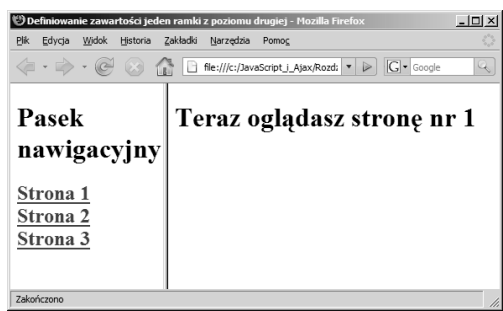
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Strict//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Pasek nawigacyjny</title>
  <script type="text/javascript"
  ➥src="script04.js"></script>
</head>
<body>
  <h1>Pasek nawigacyjny</h1>
  <h2>
  <a href="frame4a.html">Strona 1</a><br />
  <a href="frame4b.html">Strona 2</a><br />
  <a href="frame4c.html">Strona 3</a>
  </h2>
</body>
</html>
```

Skrypt 5.8. *Ten prosty kod JavaScript całkowicie wystarczy do wykonania tego zadania*

Skrypt

```
window.onload = initLinks;

function initLinks() {
  for (var i=0; i<document.links.
  ➥length; i++) {
    document.links[i].target = "content";
  }
}
```



Rysunek 5.8. *Oto ramka służąca za pasek nawigacyjny oraz strona z treścią wyświetlana w osobnej ramce*
```

Skrypt 5.9. *Ten zestaw ramek ładuje podstawową stronę, która zostanie zastąpiona przez stronę wygenerowaną w JavaScripcie*

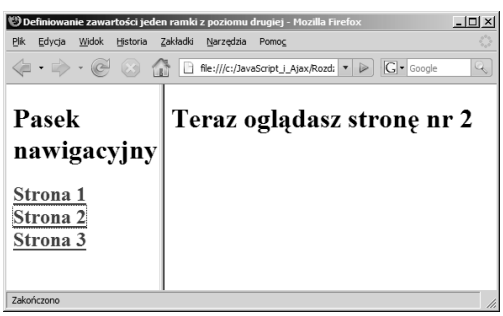
Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Frameset//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ładowanie jednej ramki z poziomu
  ➥drugiej</title>
</head>
<frameset cols="30%,70%">
  <frame src="left5.html" name="left"
  ➥id="left" />
  <frame src="frame5.html" name="content"
  ➥id="content" />
</frameset>
</html>
```

Skrypt 5.10. *A oto nasza podstawowa strona*

Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ramka z treścią</title>
</head>
<body bgcolor="#FFFFFF">
</body>
</html>
```



Rysunek 5.9. *Oto wynik działania skryptu 5.8 — strona WWW przygotowana w języku JavaScript. Zawartość prawej ramki została wygenerowana przez skrypt JavaScriptu*

Tworzenie i ładowanie ramek dynamicznych

Za pomocą JavaScriptu można dynamicznie generować zawartość strony. Okazuje się to przydatne przy ładowaniu do ramek dynamicznych danych uzależnionych od wyboru dokonanego w innej ramce. Skrypt 5.9 tworzy zestaw ramek i ładuje do głównej ramki podstawową stronę z treścią (skrypt 5.10), natomiast opisywany za chwilę skrypt 5.11 umożliwia dynamiczne przygotowanie różnych treści i załadowanie ich do głównej ramki. Powstaje w ten sposób strona przedstawiona na rysunku 5.9.

Skrypt 5.11. *Ten skrypt tworzy stronę i ładuje ją do ramki przeznaczonej na treść*

Skrypt

```
window.onload = initLinks;

function initLinks() {
  for (var i=0; i<document.links.length; i++) {
    document.links[i].onclick = writeContent;
    document.links[i].thisPage = i+1;
  }
}

function writeContent() {
  var newText = "<h1>Oglądasz właśnie stronę
  ➥nr " + this.thisPage + ".<\/h1>";

  var contentWin = parent.document.getElementById
  ➥("content").contentWindow;
  contentWin.document.body.innerHTML = newText;
  return false;
}
```

Aby dynamicznie załadować stronę do ramki z innej ramki:

1.
```
function initLinks() {
  for (var i=0; i<document.links.
  ➥length; i++) {
    document.links[i].onclick =
    ➥writeContent;
    document.links[i].thisPage = i+1;
  }
```

   Funkcja `initLinks()` zaczyna się tak samo jak w skrypcie 5.8 — od przejrzenia wszystkich łączy znajdujących się na stronie. Następnie do każdego łącza dodawane są dwie rzeczy: funkcja obsługująca zdarzenie `onclick` oraz nowa właściwość `thisPage`. We właściwości tej zapisywany jest numer strony, która ma zostać wyświetlona po kliknięciu łącza. I tak, łącze numer 0 wyświetla stronę 1, łącze numer 1 — stronę 2 itd. Funkcją obsługującą zdarzenie `onclick` wszystkich łączy jest funkcja `writeContent()`.

2.
```
var newText = "<h1>Oglądasz właśnie stronę
➥nr " + this.thisPage + ".<\/h1>";

var contentWin = parent.document.
➥getElementById("content").contentWindow;
contentWin.document.body.innerHTML =
➥newText;
```

   Oto najważniejsza część funkcji `writeContent()`. Najpierw deklarowana jest zmienna `newText` i przypisywany jest jej jakiś tekst. Następnie definiujemy zmienną `contentWin`, postępując dokładnie tak samo jak w skrypcie 5.6, a na koniec właściwości `contentWin.document.body.innerHTML` przypisujemy wartość zmiennej `newText`. W ramach wyjaśnienia: w zmiennej `contentWin` zapisujemy dokument zapisany w ramce, wybieramy z niego element body, a następnie modyfikujemy właściwość `innerHTML` tego elementu.

3. `return false;`

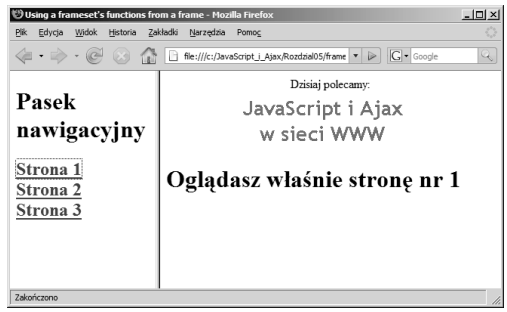
   Na koniec funkcja `writeContent()` zwraca wartość fałszu, która zakazuje przeglądarce ładowania dokumentu wskazywanego przez atrybut `href` kliknięte łącza. Bez tej instrukcji przeglądarka zastąpiłaby wygenerowaną właśnie treść stroną wskazywaną przez łącze, a to nie jest już potrzebne.

Wskazówki

- Dlaczego w kroku drugim przed znakiem ukośnika (/) został umieszczony znak lewego ukośnika (\)? Zgodnie ze specyfikacją języka HTML przeglądarka może zinterpretować początek znacznika zamykającego (</) jako znak końca wiersza. Lewy ukośnik powoduje wyróżnienie ukośnika, dzięki czemu kod HTML może zostać prawidłowo zinterpretowany.
- Korzystamy tu z metody `parent.document.getElement...`, a nie `document.getElement...`, której użyliśmy w skrypcie 5.6. Po prostu w tamtym skrypcie kod JavaScript był wywoływany przez stronę *zestawu ramek*, a tutaj wywoływany jest przez jedną ze stron *umieszczonych w ramkach*. W takiej sytuacji musimy przejść w hierarchii o jeden poziom wyżej (do strony zestawu ramek) i dopiero tam zainteresować się stroną z treścią (strona potomna strony zestawu ramek).

Funkcje wspólne dla kilku ramek

Często stosowany jest układ ramek, w którym jedna z nich służy do nawigacji, a druga zawiera właściwą treść, która może być podzielona na wiele stron. Jeśli wszystkie strony z treścią korzystają z identycznego kodu JavaScriptu, rozsądne będzie przeniesienie wywołania pliku z kodem JavaScriptu na stronę, która jest zawsze wyświetlana, zamiast przepisywania go na wszystkich możliwych stronach z treścią. Na rysunku 5.10 widać, jak wykorzystaliśmy tę opcję do udostępnienia wielu stronom funkcji, która zwraca losowo wybrany baner. Definicję układu ramek przedstawiliśmy w skrypcie 5.12.



Rysunek 5.10. *Informacje po prawej stronie tworzone są przez kod wywoływany z zestawu ramek*

Aby skorzystać z funkcji znajdującej się na innej stronie:

1. `var bannerArray = new Array("images/redBanner.gif", "images/greenBanner.gif", "images/blueBanner.gif");`

   Zaczynamy od utworzenia tablicy zawierającej nazwy wszystkich banerów i zapisania jej do zmiennej `bannerArray`.

2. `window.onload = initFrames;`

   Po załadowaniu strony zostanie wywołana funkcja `initFrames()`.

3. `var leftWin = document.getElementById("left").contentWindow.document;`

   Teraz zaczynamy analizować kod funkcji `initFrames()`. Na początku tworzona jest zmienna `leftWin`, której przypisujemy wartość w ten sam sposób co w poprzednich przykładach: pobieramy element na podstawie nazwy ramki (`document.getElementbyId(left)`), wybieramy z niego właściwość `contentWindow`, a z niej z kolei pobieramy obiekt `document`.

Skrypt 5.12. *Ten skrypt pozwala na współdzielenie funkcji przez kilka ramek*

Skrypt

```
var bannerArray = new Array("images/
➥redBanner.gif", "images/greenBanner.gif",
➥"images/blueBanner.gif");

window.onload = initFrames;

function initFrames() {
  var leftWin = document.getElementById
  ➥("left").contentWindow.document;

  for (var i=0; i<leftWin.links.length; i++)
{
    leftWin.links[i].target = "content";
    leftWin.links[i].onclick = resetBanner;
  }

  setBanner();
}

function setBanner() {
  var contentWin = document.getElementById
  ➥("content").contentWindow.document;
  var randomNum = Math.floor(Math.random()
  ➥* bannerArray.length);

  contentWin.getElementById("adBanner").
  ➥src = bannerArray[randomNum];
}

function resetBanner() {
  setTimeout("setBanner()",1000);
}
```

4. `for (var i=0; i<leftWin.links.length; ➥i++) { leftWin.links[i].target = "content"; leftWin.links[i].onclick = resetBanner;`

   Funkcja jest wywoływana w kontekście zestawu ramek, dlatego konfigurowanie łączy ze strony nawigacyjnej wygląda nieco inaczej niż w poprzednich przykładach. Tym razem dla każdego łącza modyfikujemy zarówno właściwość `target` (przypisujemy jej wartość `content`), jak i funkcję obsługi zdarzenia `onclick` (będzie ono obsługiwane przez funkcję `resetBanner()`).

5. `setBanner();`

   Ostatni krok inicjacji, czyli wywołanie funkcji `setBanner()`.

6. `var contentWin = document.getElementById ➥("content").contentWindow.document;`

   Funkcja `setBanner()` ładuje zawartość okna z treścią i generuje liczbę losową. Następnie znajdujący się na stronie baner reklamowy zastępowany jest banerem losowo wybranym z tablicy. Zaczynamy od tworzenia zmiennej `contentWin` i przypisania jej wartości w dokładnie taki sam sposób jak w poprzednich przykładach: na podstawie nazwy ramki (`content`) pobieramy odpowiedni element (`document.getElementById ("content")`), z elementu pobieramy właściwość `contentWindow`, a z niej — interesujący na dokument.

7. `var randomNum = Math.floor(Math.random() * ➥bannerArray.length);`

   W tym wierszu wykorzystujemy funkcję `Math.random()` i jej wynik mnożymy przez liczbę elementów zapisanych w tablicy `bannerArray`. W ten sposób otrzymujemy liczbę losową z zakresu od 0 do liczby elementów w tablicy. Następnie wynik jest umieszczany w zmiennej `randomNum`.

8. `contentWin.getElementById("adBanner").src = ➥bannerArray[randomNum];`

   Właściwości `src` elementu `adBanner` przypisujemy wartość wylosowanego elementu tablicy. Jest to nazwa nowego obrazka, który zostanie teraz wyświetlony na stronie.

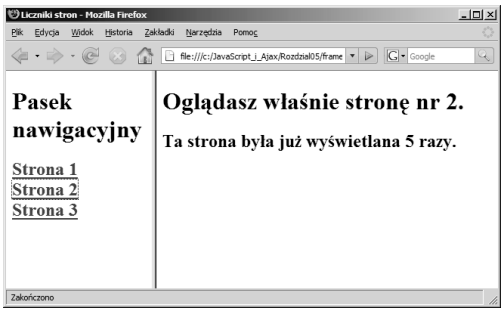
9. ```
   function resetBanner() {
     setTimeout("setBanner()",1000);
   }
   ```

   Funkcja `resetBanner()` ma tylko jeden wiersz kodu, mimo to jej działanie może nie być do końca jasne. Funkcja czeka na załadowanie całej treści w ramce `content` (jedna sekunda powinna wystarczyć), a następnie wywołuje funkcje `setBanner()` w celu wybrania nowego banera.

   Jeżeli funkcja `setBanner()` zostałaby wywołana od razu, to mogłoby się okazać, że nowa treść ramki z `content` nie została jeszcze załadowana. W takiej sytuacji na pewno pojawiłyby się problemy polegające albo na wystąpieniu błędu (nie znaleziono elementu `adBanner`), albo na dokonaniu podmiany banera na stronie, która jest właśnie usuwana z ramki.

Wskazówka

- Proszę zauważyć, że funkcja `resetBanner()` *nie* zwraca wartości fałszu. Oznacza to, że przeglądarka wykona zapisane w funkcji zadania i załaduje stronę wskazywaną przez atrybut `href`. To właśnie dlatego w skrypcie definiowana jest funkcja obsługi zdarzenia `onclick` i wartość właściwości `target`.
   ```



Rysunek 5.11. *Kod JavaScript informuje nas, że stronę 2 odwiedziliśmy już kilka razy*

Skrypt 5.13. *Zawartość znacznika zmieniana jest przez kod JavaScript*

Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ramka z treścią</title>
<body bgcolor="#FFFFFF">
  <h1>Oglądasz właśnie stronę nr 1.</h1>
  <h2>Ta strona była już wyświetlana
  ➥<span id="pageCt"> </span> razy.</h2>
</body>
</html>
```

Przechowywanie informacji w ramkach

W sytuacjach podobnych do opisywanego wcześniej przykładu można też w wyświetlanej na stałe ramce przechowywać informacje wykorzystywane przez inne ramki. Strona z rysunku 5.11 pokazuje, ile razy użytkownik odwiedzał daną stronę w ramach jednej sesji. Kod jednej z trzech (niemal identycznych) stron z treścią prezentujemy w skrypcie 5.13. Tutaj również zewnętrzny kod JavaScript (przedstawiony w skrypcie 5.14) jest wywoływany przez stronę zestawu ramek.

Aby zliczać wyświetlenia strony:

1. `<h2>Ta strona była już wyświetlana ➥<span id="pageCt"> </span> razy.</h2>`

   W skrypcie 5.13 znajduje się znacznik `<span>` oznaczony specjalnym identyfikatorem. Jak widać w samym znaczniku nie ma żadnej treści, ale możemy ją dopisać za pomocą JavaScriptu. Musieliśmy umieścić w nim choć jedną spację, ponieważ niektóre przeglądarki stwierdzają, że znacznik jest pusty, i w związku z tym w ogóle go nie tworzą. Jeżeli coś ma się stać elementem dokumentu, to musi zawierać choćby minimalny tekst.

2. `var pageCount = new Array(0,0,0); var pageArray = new Array("frame7a.html", ➥"frame7b.html","frame7c.html");`

   Na początku skryptu 5.14 deklarujemy dwie tablice i wpisujemy do nich wartości początkowe.

3. `for (var i=0; i<leftWin.links.length; ➥i++) { leftWin.links[i].onclick = resetPageCt; leftWin.links[i].thisPage = i;`

   Funkcja `initFrames()` zaczyna się tak samo jak w poprzednich przykładach, ale tym razem zdarzenie `onclick` będzie obsługiwać funkcja `resetPageCt()`, natomiast właściwości `thisPage` nadajemy wartość zmiennej `i`. W ten sposób właściwość ta otrzyma wartości 0, 1 lub 2 w zależności od łącza.

4. `bumpPageCt(0);`

   Funkcja kończy się wywołaniem funkcji `bumpPageCt()` i przekazaniem jej wartości zerowej. Podnosi ona liczbę odwiedzin dla strony pierwszej.

Skrypt 5.14. *W tym skrypcie modyfikowana jest wartość znacznika o identyfikatorze pageCt*

Skrypt

```
var pageCount = new Array(0,0,0);
var pageArray = new Array("frame7a.html",
➥"frame7b.html","frame7c.html");

window.onload = initFrames;

function initFrames() {
  var leftWin = document.getElementById
  ➥("left").contentWindow.document;

  for (var i=0; i<leftWin.links.length; i++)
{
    leftWin.links[i].onclick = resetPageCt;
    leftWin.links[i].thisPage = i;
  }

  bumpPageCt(0);
}

function bumpPageCt(currPage) {
  pageCount[currPage]++;

  var contentWin = document.getElementById
  ➥("content").contentWindow.document;
  contentWin.getElementById("pageCt").
  ➥innerHTML = pageCount[currPage] + " ";
}

function resetPageCt() {
  document.getElementById("content").
  ➥contentWindow.location.href =
  ➥pageArray[this.thisPage];

  setTimeout("bumpPageCt
  ➥("+this.thisPage+")",1000);
  return false;
}
```

5.
```
function bumpPageCt(currPage) {
  pageCount[currPage]++;

  var contentWin = document.getElementById
  ➥("content").contentWindow.document;
  contentWin.getElementById("pageCt")
  ➥.innerHTML = pageCount[currPage] + " ";
}
```

Funkcja `bumpPageCt()` pobiera numer strony i zwiększa przypisany do niej licznik (przechowywany w tablicy `pageCount`). Ponownie pobieramy tutaj element `contentWin` (czyli główny dokument z treścią umieszczony w ramce). Następnie korzystamy z niego, aby zmienić zawartość właściwości `innerHTML` elementu `pageCt` na liczbę wyświetleń danej strony.

6.
```
function resetPageCt() {
  document.getElementById("content").
  ➥contentWindow.location.href =
  ➥pageArray[this.thisPage];

  setTimeout("bumpPageCt("+this.
  ➥thisPage+")",1000);
  return false;
}
```

Funkcja `resetPageCt()` jest połączeniem opisywanych wcześniej funkcji `resetBanner()` i `bumpPageCt()` uzupełnionych nowymi elementami. Nie chcemy, żeby przeglądarka sama zmieniała zawartość ramki z treścią, dlatego przypisujemy jej stronę na podstawie tablicy `pageArray`. Po zmianie zawartości ramki chcemy, żeby funkcja `bumpPageCt()` została wywołana dopiero *po* zakończeniu ładowania nowej strony, a to oznacza, że musimy skorzystać z metody `setTimeout()`. Na koniec zwracamy wartość fałszu, ponieważ ładowaniem strony zajmuje się skrypt JavaScript.

Ładowanie kilku ramek na raz

Zmienić zawartość ramki można za pomocą zwykłego łącza z atrybutem `target` — tak działa język HTML. Jednak aby zmienić jednym kliknięciem treść wielu ramek, potrzeba już JavaScriptu. Kod HTML przykładowych stron jest bardzo prosty, a wszystkie operacje wykonywane są w skrypcie 5.15. Dwa stany, jakie może przyjmować okno przeglądarki, przedstawione zostały na rysunkach 5.12 i 5.13.

Aby załadować wiele ramek na raz:

1. ```
   function initFrames() {
     var leftWin = document.getElementById
     ➥("left").contentWindow.document;

     for (var i=0; i<leftWin.links.length;
     ➥i++) {
       leftWin.links[i].onclick = setFrames;
     }
   }
   ```

   Funkcja `initFrames()` jest bardzo podobna do tych, które widzieliśmy w poprzednich przykładach. Wywoływana jest w ramach zdarzenia `onload` i p rzegląda wszystkie łącza z ramki nawigacyjnej (tej z lewej strony). Tym razem do obsługi zdarzeń `onclick` wybrana została funkcja `setFrames()`.

Skrypt 5.15. *Kod JavaScript pozwala na załadowanie wielu ramek po jednym kliknięciu użytkownika*

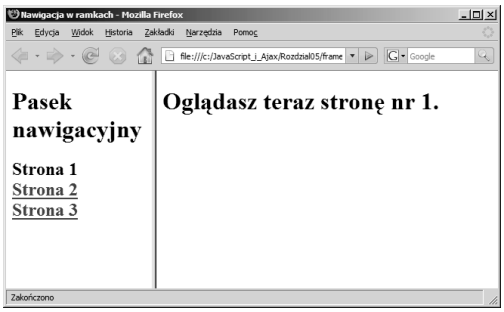
Skrypt

```
window.onload = initFrames;

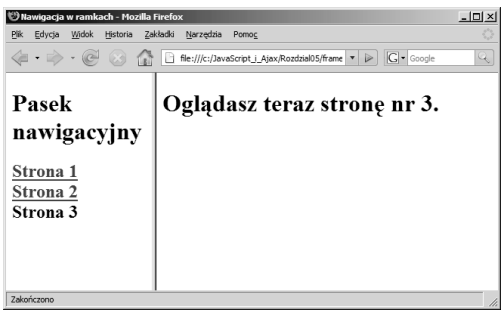
function initFrames() {
  var leftWin = document.getElementById
  ➥("left").contentWindow.document;

  for (var i=0; i<leftWin.links.length;
  ➥i++) {
    leftWin.links[i].onclick = setFrames;
  }
}

function setFrames() {
  document.getElementById("left").
  ➥contentWindow.document.location.href =
  ➥this.id + ".html";
  document.getElementById("content").
  ➥contentWindow.document.location.href =
  ➥this.href;
  setTimeout("initFrames();",1000);
  return false;
}
```
```



Rysunek 5.12. *Początkowo strona wygląda tak*



Rysunek 5.13. *Po kliknięciu jednego z łączy zmianie ulegają obie ramki*

2. `document.getElementById("left").`
`➥contentWindow.document.location.href =`
`➥this.id + ".html";`
`document.getElementById("content").`
`➥contentWindow.document.location.href =`
`➥this.href;`
`setTimeout("initFrames();",1000);`

Funkcja `setFrames()` na początku zmienia zawartość lewej ramki, korzystając przy tym z identyfikatora kliknięcia łącza. Następnie funkcja zmienia zawartość ramki z treścią, tym razem korzystając z atrybutu `href` kliknięcia łącza. Mamy dostęp do tych danych z poziomu skryptu JavaScript, więc dlaczego mielibyśmy z nich nie skorzystać?

Następnie funkcja odczekuje sekundę (ponownie korzystamy z metody `setTimeout()`), po czym wywołuje funkcję `initFrames()`. Jest to konieczne, ponieważ to właśnie w niej funkcje obsługi zdarzenia `onclick` przypisywane są do poszczególnych łączy. Niestety, po ponownym załadowaniu strony do ramki nawigacyjnej wszystkie wcześniejsze przypisania przestają obowiązywać. Jeżeli całość ma działać tak samo jak na początku, to operacje związane z inicjacją trzeba wykonać po każdym załadowaniu strony ramki nawigacyjnej.

3. `return false;`

Na koniec zwracamy wartość fałszu, tak żeby przeglądarka nie musiała się zajmować ładowaniem nowej strony.

Wskazówka

- Jeżeli będziemy szybko klikać łącza z paska nawigacyjnego, to natkniemy się na dość nieoczekiwane zachowanie: strona z treścią zostanie załadowana do ramki paska nawigacyjnego. Po prostu strona nawigacyjna musi załadować się w całości i dopiero wtedy może przyjmować kliknięcia.

Praca z elementami iframe

Element *iframe* jest elementem wierszowym, czyli ramką, która może zostać włączona do zwykłej strony HTML, i nie musi być częścią zestawu ramek. Podobnie jak w przypadku zwyczajnej ramki, element `iframe` jest całkowicie niezależnym dokumentem HTML. Na elemencie tym mogą pracować powiązane ze stroną skrypty, a zatem można za ich pomocą dynamicznie generować zawartość takiej ramki, bez konieczności stosowania całego zestawu ramek.

W tym przykładzie jedną ze stron HTML umieścimy w części wydzielonej dla treści, która zostanie zamknięta w elemencie `iframe`. Zawartość tego elementu będzie generowana za pomocą JavaScriptu. Skrypt 5.16 tworzy główną stronę wyświetlaną w przeglądarce. Skrypt 5.17 jest tylko tymczasowym dokumentem HTML ładowanym do elementu `iframe`. Wyświetla on tylko komunikat „Proszę załadować stronę", który można zobaczyć na rysunku 5.14. Zawartość strony przygotowywana jest przez kod zaprezentowany w skrypcie 5.18. W momencie gdy użytkownik kliknie jedno z łączy, kod JavaScript wpisze do elementu `iframe` nową treść strony. W tym przypadku wyświetlana jest tylko nazwa strony oraz liczba odwiedzin strony w ramach jednej sesji. Jak widać, większa część tego skryptu została zbudowana z kodu, który wykorzystywaliśmy już w poprzednich przykładach.

Skrypt 5.16. *Ta strona tworzy ramkę typu iframe i wywołuje zewnętrzny kod JavaScript*

Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ramki iframe</title>
  <script language="javascript"
  ➥type="text/javascript"
  ➥src="script09.js"></script>
</head>
<body bgcolor="#FFFFFF">
  <iframe src="frame9b.html" width="550"
  ➥height="300" name="content" id="content"
  ➥align="right" frameborder="1">Twoja
  ➥przeglądarka nie obsługuje ramek Iframe
  ➥</iframe>
  <h1>pasek nawigacyjny</h1>
  <h2>
  <a href="page1.html">Strona 1</a><br />
  <a href="page2.html">Strona 2</a><br />
  <a href="page3.html">Strona 3</a>
  </h2>
</body>
</html>
```

Skrypt 5.17. *Początkowa strona umieszczana w ramce iframe*

Skrypt

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ramka z treścią</title>
</head>
<body bgcolor="#FFFFFF">
  Proszę załadować stronę
</body>
</html>
```

Skrypt 5.18. *Ten skrypt generuje zawartość ramki iframe i zapisuje ją do odpowiedniego okna*

Skrypt

```
var pageCount = new Array(0,0,0,0);

window.onload = initFrames;

function initFrames() {
  for (var i=0; i<document.links.length;
  ➥i++) {
    document.links[i].onclick =
    ➥writeContent;
    document.links[i].thisPage = i+1;
  }
}

function writeContent() {
  pageCount[this.thisPage]++;

  var newText = "<h1>Oglądasz właśnie stronę
  ➥nr " + this.thisPage;
  newText += ".<br \/>Strona ta została
  ➥wyświetlona już ";
  newText += pageCount[this.thisPage]
  ➥+ " razy.<\/h1>";

  var contentWin =
document.getElementById("content").
➥contentWindow.document;
  contentWin.body.innerHTML = newText;
  return false;
}
```

Rysunek 5.14. *W znajdującej się po prawej stronie ramce iframe początkowo wyświetlana jest tylko prośba o kliknięcie jednego z łączy*

Aby przygotować zawartość ramki iframe:

1. `<iframe src="frame9b.html" width="550" ➥height="300" name="content" id="content" ➥align="right" frameborder="1">Twoja ➥przeglądarka nie obsługuje ramek Iframe ➥</iframe>`

   W skrypcie 5.16 znacznik `<iframe>` informuje przeglądarkę, że początkowo dokumentem wyświetlanym w ramce będzie plik *frame9b.html*. Wewnątrz znacznika zapisany jest też komunikat, który będzie wyświetlany przez przeglądarki nieobsługujące ramek `iframe`.

2. `function writeContent() {`

   Tworzymy funkcję o nazwie `writeContent`, której używać będziemy do tworzenia treści umieszczanych w elemencie `iframe`.

3. `pageCount[this.thisPage]++;`

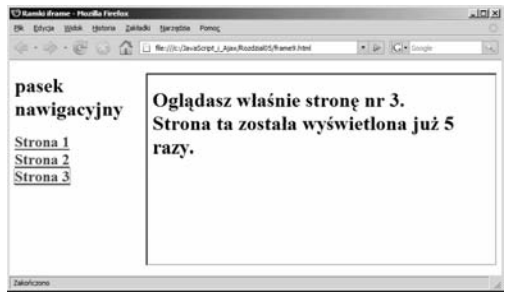
   Ten wiersz zwiększa licznik w tablicy `pageCount`, dzięki czemu wiadomo, ile razy odwiedziliśmy poszczególne strony.

4. `var newText = "<h1>Oglądasz właśnie stronę ➥nr " + this.thisPage; newText += ".<br \/>Strona ta została ➥wyświetlona już "; newText += pageCount[this.thisPage] ➥+ " razy.<\/h1>";`

   Te wiersze tworzą treść, która jest wpisywana do ramki `iframe`.

5. `var contentWin = document.getElementById ➥("content").contentWindow.document; contentWin.body.innerHTML = newText; return false;`

   Podobnie jak w poprzednich przykładach, pobieramy element `contentWin` i modyfikujemy właściwość `innerHTML` znajdującego się w nim znacznika `body`. W ramach tej zmiany wpisujemy dwa wiersze tekstu, które można zobaczyć na rysunku 5.15. To wszystko, co chcemy wykonać w tej funkcji, więc na koniec zwracamy wartość fałszu, żeby zwolnić przeglądarkę z obowiązku ładowania strony.



Rysunek 5.15. *Przy każdym kliknięciu łącza na pasku nawigacyjnym zmieniana jest zawartość ramki iframe*

Wskazówka

- Proszę zauważyć, że tym razem do ramki zapisujemy dane z poziomu strony niebędącej w zestawie ramek i dlatego możemy skorzystać z instrukcji `document.getElementById("content")`, a nie `parent.document.getElementById ("content")`, tak jak w poprzednich przykładach. Po prostu ramka `iframe` jest elementem potomnym głównej strony, dzięki czemu możemy się do niego odwoływać bezpośrednio.

Skrypt 5.19. *Ten skrypt ładuje strony HTML do ramki iframe*

Skrypt

```
window.onload = initiFrame;

function initiFrame() {
  for (var i=0; i<document.links.length;
  ➥i++) {
    document.links[i].target = "content";
    document.links[i].onclick = setiFrame;
  }
}

function setiFrame() {
  document.getElementById("content").
  ➥contentWindow.document.location.href =
  ➥this.href;
  return false;
}
```

Rysunek 5.16. *Zawartość ramki iframe jest ładowana po kliknięciu łącza z paska nawigacyjnego.*

Ładowanie ramek iframe za pomocą JavaScriptu

Oczywiście, nie mamy obowiązku wypełniania ramek `iframe` za pomocą skryptów JavaScript. Równie dobrze możemy ładować do nich osobne dokumenty HTML. W tym przykładzie pokażemy sposób takiego postępowania. Ponownie tworzymy stronę główną zawierającą ramkę `iframe` (praktycznie dokładnie taką samą jak ta ze skryptu 5.16). Oczywiście, przygotować sobie musimy też stronę z początkową zawartością ramki, podobną do tej ze skryptu 5.17. Oprócz tego tworzymy też trzy niezależne strony HTML (ich kod nie będzie prezentowany), które będą ładowane do ramki `iframe` za pomocą skryptu 5.19.

Aby załadować ramkę iframe za pomocą JavaScriptu:

1.
```
function setiFrame() {
  document.getElementById("content").
  ➥contentWindow.document.location.href =
  ➥this.href;
  return false;
}
```

   Podobnie jak w skrypcie 5.12 definiujemy we wszystkich łączach właściwość `target` oraz funkcje obsługi zdarzenia `onclick`. W tym przykładzie kliknięcie dowolnego łącza wywołuje funkcję `setiFrame()`, która ładuje nową stronę do ramki `iframe` (rysunek 5.16).