JAVASCRIPT i JQUERY

Interaktywne strony WWW dla każdego

Podręcznik Front-End Developera

JON DUCKETT

Tytuł oryginału: JavaScript and JQuery: Interactive Front-End Web Development

Tłumaczenie: Robert Górczyński

ISBN: 978-83-8322-755-9

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

Printed in Poland.

SPIS TREŚCI

Wprowadzenie		5
Rozdział 1.	ABC programowania	17
Rozdział 2.	Podstawowe instrukcje JavaScript	59
Rozdział 3.	Funkcje, metody i obiekty	91
Rozdział 4.	Decyzje i pętle	151
Rozdział 5.	Obiektowy model dokumentu	189
Rozdział 6.	Zdarzenia	249
Rozdział 7.	jQuery	299
Rozdział 8.	Ajax i JSON	373
Rozdział 9.	API	415
Rozdział 10.	Obsługa błędów i debugowanie	455
Rozdział 11.	Panele zawartości	493
Rozdział 12.	Filtrowanie, wyszukiwanie i sortowanie	533
Rozdział 13.	Usprawnienia i weryfikacja formularzy sieciowych	573
Skorowidz		635

8

AJAX I JSON

Ajax to technika wczytywania danych we fragmencie strony bez potrzeby odświeżania jej całości. Dane są najczęściej dostarczane w formacie o nazwie JSON (ang. *JavaScript Object Notation*).

Możliwość wczytania nowej zawartości tylko na części strony znacznie poprawia wrażenia użytkownika, ponieważ nie musi on oczekiwać na odświeżenie całej strony w celu uaktualnienia tylko jej fragmentu. Doprowadziło to do powstania tak zwanych aplikacji internetowych w postaci pojedynczych stron (narzędzia internetowe przypominające w działaniu tradycyjne oprogramowanie, choć uruchamiane w przeglądarce internetowej). W tym rozdziale zostaną omówione następujące zagadnienia:

CO TO JEST AJAX?

Ajax pozwala na żądanie danych z serwera oraz wczytanie ich bez konieczności odświeżenia całej strony.

FORMATY DANYCH

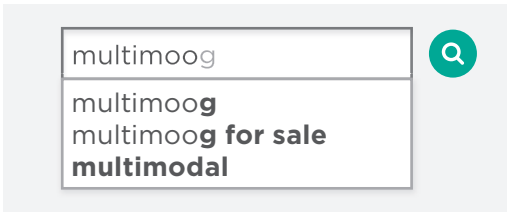
Serwery zwykle wysyłają dane w formatach HTML, XML i JSON — formaty te poznasz w tym rozdziale.

JQUERY I AJAX

jQuery ułatwia wykonywanie żądań Ajax oraz przetwarzanie danych zwróconych przez serwer.

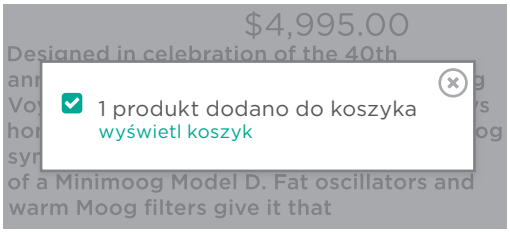
CO TO JEST AJAX?

Z technologią Ajax mogłeś się zetknąć w wielu witrynach internetowych, nawet jeśli nie wiedziałeś o jej zastosowaniu.

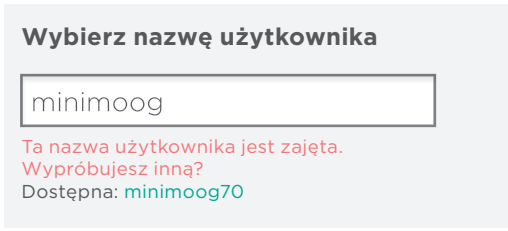


Wyszukiwanie na żywo (lub automatyczne uzupełnianie) najczęściej opiera się na technologii Ajax. Funkcję takiego wyszukiwania oferuje między innymi Google. Gdy wprowadzasz wyrażenia w polu wyszukiwania na stronie głównej Google, czasami otrzymujesz wyniki, zanim zakończysz wpisywanie tekstu.

Witryny internetowe zawierające treść generowaną przez użytkowników, na przykład Twitter i Flickr, mogą pozwalać na wyświetlanie tej treści (takich jak najnowszy tweet lub zdjęcia) na innych witrynach, co wiąże się z pobieraniem informacji z serwerów wymienionych usług.



Czasami podczas zakupów internetowych dodajesz produkt do koszyka, który zostaje uaktualniony bez opuszczania bieżącej strony. Jednocześnie witryna może wyświetlić komunikat potwierdzający dodanie produktu do koszyka.



Gdy rejestrujesz konto użytkownika w witrynie, zanim ukończysz wypełnianie całego formularza sieciowego, skrypt może sprawdzić, czy wybrana nazwa użytkownika jest dostępna.

Witryny internetowe mogą korzystać z technologii Ajax do wczytywania danych w tle, aby mogły być użyte lub wyświetlone później.

DLACZEGO UŻYWAĆ TECHNOLOGII AJAX?

Ajax wykorzystuje model przetwarzania asynchronicznego. Oznacza to, że użytkownik może wykonywać inne zadania, gdy przeglądarka internetowa oczekuje na wczytanie danych. Dzięki temu użytkownik ma lepsze wrażenia.

UŻYCIE TECHNOLOGII AJAX PODCZAS WCZYTYWANIA STRON

Kiedy przeglądarka napotyka znacznik `<script>`, najczęściej zatrzymuje generowanie pozostałej części strony aż do chwili wczytania i przetworzenia danego skryptu. Takie podejście jest nazywane **modelem przetwarzania synchronicznego**.

Kiedy podczas wczytywania strony skrypt musi pobrać dane z serwera (na przykład kursy wymiany walut lub informacje o stanie), przeglądarka nie tylko czeka na wczytanie i przetworzenie skryptu, ale również na pobranie z serwera danych, które mają być wyświetlone przez skrypt.

W przypadku technologii Ajax przeglądarka może żądać pewnych danych z serwera, a następnie (już po wykonaniu żądania danych) kontynuować wczytywanie pozostałej części strony i przetwarzanie działań podejmowanych przez użytkownika na tej stronie. Takie podejście jest nazywane **modelem przetwarzania asynchronicznego** (inaczej **nieblokującego**).

Przeglądarka internetowa nie czeka na pobranie zewnętrznych danych, aby wyświetlić stronę. Kiedy serwer udzieli odpowiedzi i dostarczy dane, następuje wywołanie zdarzenia (podobnie jak zdarzenie `load` jest wywoływane po zakończeniu wczytywania strony). Zdarzenie to może wywołać funkcję odpowiedzialną za przetworzenie danych.

UŻYCIE TECHNOLOGII AJAX PO WCZYTANIU STRON

Po wczytaniu strony, jeżeli chcesz uaktualnić zawartość widzianą przez użytkownika w oknie przeglądarki, to zwykle odświeżasz całą stronę. Oznacza to, że użytkownik musi zaczekać na pobranie całej zupełnie nowej strony i wygenerowanie jej przez przeglądarkę.

Gdy masz do dyspozycji technologię Ajax i chcesz uaktualnić tylko *fragment* strony, wystarczy uaktualnić zawartość jednego elementu. Odbywa się to przez przechwycenie zdarzenia (na przykład klika łącze lub wysyła formularz sieciowy), a następnie żądanie nowej zawartości z serwera za pośrednictwem żądania asynchronicznego.

Podczas wczytywania danych użytkownik może kontynuować pracę z pozostałą częścią strony internetowej. Gdy serwer udzieli odpowiedzi, specjalne żądanie Ajax powoduje wywołanie innej części skryptu, odczytującej nowe dane z serwera i uaktualniającej po prostu jedną część strony.

Ponieważ nie ma potrzeby odświeżania całej strony, dane są wczytywane szybciej, a użytkownik nadal może korzystać z pozostałej części strony, oczekując na wczytanie danych.

Na początku Ajax był akronimem oznaczającym technologie używane podczas wykonywania żądań asynchronicznych, takich jak omówione powyżej — asynchroniczny JavaScript i XML. Od tamtego czasu wiele się zmieniło, technologie są rozwijane, a pojęcie Ajax oznacza teraz *grupę* technologii oferujących asynchroniczną funkcjonalność w przeglądarce internetowej.

JAK DZIAŁA AJAX?

Podczas użycia technologii Ajax przeglądarka internetowa wykonuje żądanie dotyczące pewnych informacji z serwera WWW. Następnie przetwarza odpowiedź udzieloną przez serwer i wyświetla ją na stronie.

1. ŻĄDANIE

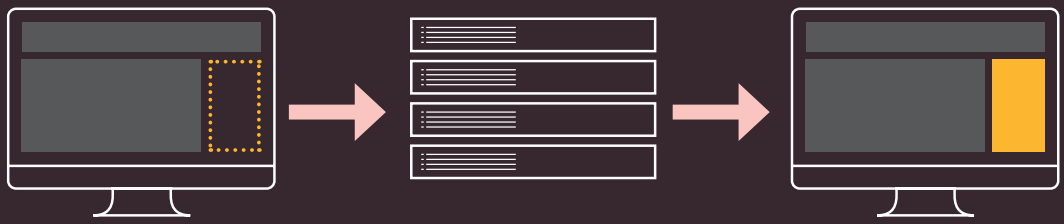
Przeglądarka żąda danych z serwera WWW.

W SERWERZE WWW

Serwer WWW udziela odpowiedzi, dostarczając żądane dane (najczęściej w formacie HTML, XML lub JSON).

2. ODPOWIEDŹ

Przeglądarka przetwarza zawartość i umieszcza ją na stronie.



Przeglądarka wysyła do serwera WWW żądanie pewnych danych. Żądanie może zawierać informacje wymagane przez serwer; podobnie formularz sieciowy wysyła dane do serwera.

W przeglądarkach implementowany jest obiekt o nazwie `XMLHttpRequest` przeznaczony do obsługi żądań Ajax. Po wysłaniu żądania przeglądarka nie musi czekać na udzielenie odpowiedzi przez serwer.

Operacje przeprowadzane w serwerze nie są częścią tego, co określamy mianem Ajax.

Technologie działające po stronie serwera, na przykład ASP.NET, PHP, Node.js i Ruby, mogą generować strony internetowe dla użytkowników. Kiedy pojawia się żądanie Ajax, serwer może udzielić odpowiedzi w postaci danych HTML lub też w innym formacie, takim jak JSON lub XML (który przeglądarka konwertuje na postać HTML).

Kiedy serwer zakończy udzielanie odpowiedzi na żądanie, przeglądarka wywołuje zdarzenie (podobnie jak może wywołać zdarzenie po zakończeniu wczytywania strony).

Zdarzenie to można wykorzystać do uruchomienia funkcji JavaScript przetwarzającej dane i umieszczającej je na stronie (bez wpływu na pozostałą część strony).

OBSŁUGA ŻĄDAŃ I ODPOWIEDZI AJAX

W celu wykonania żądania Ajax przeglądarka internetowa używa obiektu `XMLHttpRequest`. Po otrzymaniu z serwera odpowiedzi na żądanie ten sam obiekt `XMLHttpRequest` będzie przetwarzał wynik.

ŻĄDANIE

```
① var xhr = new XMLHttpRequest();
② xhr.open('GET', 'data/test.json', true);
③ xhr.send('search=arduino');
```

1. Utworzenie egzemplarza obiektu `XMLHttpRequest` za pomocą notacji konstruktora obiektu (tę notację poznałeś w rozdziale 3., w podrozdziale „Wartości truthy i falsy"). Użyte zostało słowo kluczowe `new`, a obiekt jest przechowywany w zmiennej, której nazwa `xhr` jest skrótem od **X**ML**H**ttp**R**equest (nazwa obiektu).

2. Metoda `open()` obiektu `XMLHttpRequest` przygotowuje żądanie. Ma ona trzy parametry, które poznasz w podrozdziale „Wczytywanie HTML za pomocą technologii Ajax":
i) nazwa metody HTTP;
ii) adres URL strony, która będzie obsługiwać żądanie;
iii) wartość boolowska, wskazująca, czy żądanie powinno być asynchroniczne.

3. Metoda `send()` jest odpowiedzialna za wysłanie do serwera przygotowanego wcześniej żądania. Informacje dodatkowe można przekazać serwerowi, umieszczając je w nawiasie. Jeżeli nie są przekazywane żadne informacje dodatkowe, można się spotkać z użyciem słowa kluczowego `null` (choć to nie jest ściśle wymagane): `xhr.send(null)`.

ODPOWIEDŹ

```
① xhr.onload = function() {
② if (xhr.status === 200) {
    // Kod odpowiedzialny za przetwarzanie odpowiedzi udzielonej przez serwer.
  }
}
```

1. Kiedy przeglądarka otrzyma i wczyta odpowiedź z serwera, nastąpi wywołanie zdarzenia `onload`. To z kolei spowoduje wykonanie funkcji (tutaj jest to funkcja anonimowa).

2. Funkcja sprawdza stan właściwości `status` obiektu. Ma to na celu upewnienie się, że odpowiedź otrzymana z serwera jest prawidłowa. (Jeżeli wymieniona właściwość jest pusta, należy sprawdzić konfigurację serwera).

Warto zwrócić uwagę, że IE9 to pierwsza wersja przeglądarki Internet Explorer, która obsługuje ten sposób odpowiedzi na żądania Ajax. Jeżeli chcesz, aby to funkcjonowało także w starszych przeglądarkach, możesz użyć biblioteki jQuery (patrz podrozdział „jQuery i Ajax — żądania").

FORMATY DANYCH

Odpowiedź na żądanie Ajax jest zwykle udzielana w jednym z trzech następujących formatów: HTML, XML lub JSON. Poniżej przedstawiono porównanie wymienionych formatów. Wprowadzenie do XML i JSON znajdziesz na trzech kolejnych stronach.

HTML

Prawdopodobnie najlepiej znasz format HTML i dlatego podczas uaktualniania fragmentu strony użycie tego formatu jest najłatwiejszym sposobem umieszczenia nowych danych na stronie.

ZALETY

- Łatwe tworzenie, żądanie i wyświetlanie danych.
- Dane otrzymane z serwera są bezpośrednio umieszczane na stronie. W przeciwieństwie do dwóch pozostałych formatów dane nie muszą być przetwarzane przez przeglądarkę.

WADY

- Serwer musi wygenerować HTML w postaci gotowej do użycia na danej stronie.
- Ten format nie sprawdza się zbyt dobrze w aplikacjach uruchamianych poza przeglądarką. Nie charakteryzuje się zbyt dobrą **przenośnością danych**.
- Żądanie musi pochodzić z tej samej domeny* (patrz uwaga poniżej).

XML

Dane XML wyglądają podobnie jak HTML, ale nazwy znaczników są inne, ponieważ opisują znajdujące się w nich dane. Składnia jest bardziej rygorystyczna niż w HTML.

ZALETY

- To elastyczny format, pozwalający na przedstawienie skomplikowanych struktur danych.
- Doskonale sprawdza się na różnych platformach i w różnych aplikacjach.
- Dane są przetwarzane za pomocą tych samych metod DOM co w przypadku HTML.

WADY

- Język ten jest uznawany za rozwlekły, ponieważ znaczniki dodają wiele dodatkowych znaków do przekazywanych danych.
- Żądanie musi pochodzić z tej samej domeny co pozostała część strony* (patrz uwaga poniżej).
- Przetworzenie wyniku może wymagać ogromnej ilości kodu.

JSON

W celu przedstawienia danych format JSON (ang. *JavaScript Object Notation*) używa składni podobnej do notacji literału obiektu, którą poznałeś w rozdziale 3., w podrozdziale „Utworzenie obiektu — notacja literału”.

ZALETY

- Możliwość wywołania z dowolnej domeny (JSON-P, CORS).
- Większa zwięzłość niż w HTML i XML.
- Powszechne wykorzystanie w języku JavaScript (zyskuje coraz większą popularność w aplikacjach sieciowych).

WADY

- Składnia musi być bezbłędna. Brakujący znak cytowania, przecinek lub dwukropek może uniemożliwić prawidłowe działanie pliku.
- Ponieważ to jest kod JavaScript, może zawierać zawartość o złośliwym działaniu (patrz ataki typu XSS w rozdziale 5., w podrozdziale „Ataki typu XSS”). Dlatego też danych w formacie JSON należy używać tylko z zaufanych źródeł.

* Przeglądarki pozwalają technologii Ajax na wczytywanie kodu HTML i XML jedynie z tej samej domeny, z której pochodzi strona. Na przykład jeśli strona znajduje się w domenie *www.przyklad.pl*, to żądanie Ajax musi zwrócić dane pochodzące z *www.przyklad.pl*.

XML — EXTENSIBLE MARKUP LANGUAGE

Kod XML wygląda jak HTML, ale znaczniki zawierają różne słowa. Celem znaczników jest opisanie rodzaju przechowywanych przez nie danych.

```
<?xml version="1.0" encoding="utf-8" ?>
<events>
  <event>
    <location>San Francisco, CA</location>
    <date>1 maja</date>
    <map>img/map-ca.png</map>
  </event>
  <event>
    <location>Austin, TX</location>
    <date>15 maja</date>
    <map>img/map-tx.png</map>
  </event>
  <event>
    <location>Nowy Jork, NY</location>
    <date>30 maja</date>
    <map>img/map-ny.png</map>
  </event>
</events>
```

Plik XML można przetwarzać z wykorzystaniem tych samych metod DOM, które są używane w pracy z HTML. Ponieważ przeglądarki w różny sposób traktują znaki odstępu w dokumentach HTML i XML, przetwarzanie XML jest łatwiejsze z zastosowaniem jQuery niż zwykłego języka JavaScript (podobnie jak w przypadku HTML).

Podobnie jak HTML jest językiem znaczników używanym do opisania struktury i semantyki strony internetowej, XML można wykorzystać do opracowania języków znaczników dla innego rodzaju danych, zupełnie dowolnych, od raportów giełdowych aż po informacje medyczne.

Znaczniki w pliku XML powinny opisywać znajdujące się w nich dane. Dlatego też nawet jeśli nigdy wcześniej nie spotkałeś się z kodem pokazanym po lewej stronie, to i tak bez problemu zauważysz, że zawiera on informacje o kilku wydarzeniach (element `<events>`; poszczególne wydarzenia są przedstawiane za pomocą oddzielnych znaczników `<event>`).

Format XML działa na każdej platformie; największą popularność zyskał na początku wieku, ponieważ znacznie ułatwiał przenoszenie danych między różnego rodzaju aplikacjami. To jednocześnie niezwykle elastyczny format danych, za jego pomocą można przedstawiać naprawdę skomplikowane struktury danych.

JAVASCRIPT OBJECT NOTATION

Dane można sformatować za pomocą JSON (wym. **'dʒeɪsən**). Składnia jest bardzo podobna do notacji literału obiektu, ale to nie jest obiekt.

Dane w formacie JSON przypominają notację literału obiektu, którą poznałeś w rozdziale 3., w podrozdziale „Utworzenie obiektu — notacja literału". To jednak nie jest obiekt, lecz dane w postaci zwykłego tekstu.

Różnica może wydawać się drobna, ale pamiętaj, że HTML to także zwykły tekst, a przeglądarka internetowa konwertuje go na postać obiektów modelu DOM.

Rzeczywistych obiektów nie można przekazywać przez sieć. Zamiast tego wysyłany jest tekst, który następnie będzie przez przeglądarkę internetową skonwertowany na postać obiektów.

```
{
  "location": "San Francisco, CA",
  "capacity": 270,
  "booking":  true
}
```

KLUCZ (ujęty w cudzysłów)

WARTOŚĆ

KLUCZ

W formacie JSON klucz powinien zostać ujęty w **cudzysłów** (a nie apostrofy).

Klucz (inaczej nazwa) jest oddzielony od wartości dwukropkiem.

Poszczególne pary klucz-wartość są rozdzielone przecinkami. Jednak zwróć uwagę na *brak* przecinka po ostatniej parze.

WARTOŚĆ

Wartość może być jednego z wymienionych poniżej typów danych (niektóre przedstawiono na przykładzie powyżej, pozostałe pokazano w przykładzie na stronie po prawej).

TYP DANYCH	OPIS
`string`	Tekst (musi być ujęty w cudzysłów).
`number`	Liczba.
`Boolean`	Wartość `true` lub `false`.
`array`	Tablica wartości — może to być również tablica obiektów.
`object`	Obiekt JavaScript — może zawierać obiekty potomne lub tablice.
`null`	Wartość pusta lub w ogóle brak wartości.

PRACA Z DANYMI JSON

Obiekt `JSON` w języku JavaScript może zamienić dane JSON na postać obiektu JavaScript. Pozwala również przeprowadzić konwersję obiektu JavaScript na ciąg tekstowy.

```
{
  "events": [
    {
      "location": "San Francisco, CA",
      "date": "1 maja",
      "map": "img/map-ca.png"
    },
    {
      "location": "Austin, TX",
      "date": "15 maja",
      "map": "img/map-tx.png"
    },
    {
      "location": "Nowy Jork, NY",
      "date": "30 maja",
      "map": "img/map-ny.png"
    }
  ]
}
```

● OBIEKT ● TABLICA

Obiekt po lewej stronie przedstawia serię trzech wydarzeń, o których informacje są przechowywane w tablicy o nazwie `events`. Tablica wykorzystuje notację nawiasu kwadratowego i przechowuje trzy obiekty (po jednym dla każdego wydarzenia).

Metoda `JSON.stringify()` przeprowadza konwersję obiektu JavaScript na ciąg tekstowy, sformatowany jako dane JSON. W ten sposób obiekty JavaScript można przesyłać z przeglądarki do innej aplikacji.

Metoda `JSON.parse()` przetwarza ciąg tekstowy zawierający dane JSON. Przeprowadza konwersję danych JSON na postać obiektów JavaScript gotowych do użycia w przeglądarce.

Obsługa w przeglądarkach: Chrome 3, Firefox 3.1, IE8 i Safari 4.

Obiekt można zapisać także w jednym wierszu, jak pokazano poniżej:

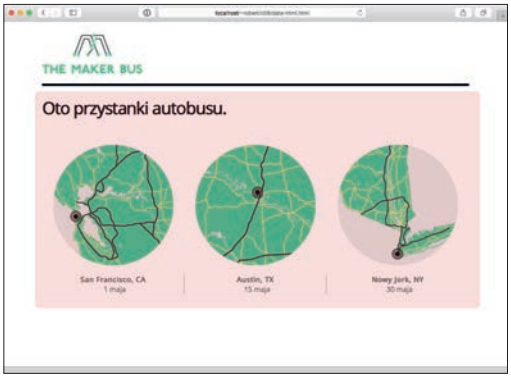
```
{
  "events": [
    { "location": "San Francisco, CA", "date": "1 maja", "map": "img/map-ca.png" },
    { "location": "Austin, TX", "date": "15 maja", "map": "img/map-tx.png" },
    { "location": "Nowy Jork, NY", "date": "30 maja", "map": "img/map-ny.png" }
  ]
}
```

WCZYTYWANIE HTML Z WYKORZYSTANIEM TECHNOLOGII AJAX

HTML to najłatwiejszy typ danych, jakie można dodać na stronie z wykorzystaniem technologii Ajax. Przeglądarka wygeneruje je dokładnie tak samo jak każde inne dane HTML.
Do nowej zawartości będą zastosowane istniejące reguły CSS.

Poniżej przedstawiono przykład wczytujący za pomocą technologii Ajax dane o trzech wydarzeniach. (W kolejnych czterech przykładach wynik będzie dokładnie taki sam).

Strona otwierana przez użytkownika nie przechowuje danych dotyczących wydarzeń (oznaczone kolorem różowym). Technologia Ajax jest wykorzystywana do wczytania danych z innego pliku.



Obszar w kolorze różowym jest wczytany z wykorzystaniem technologii Ajax

Kiedy serwer udziela odpowiedzi na dowolne żądanie, powinien przekazać komunikat o stanie, wskazujący, czy żądanie zostało ukończone. Oto przykładowe wartości komunikatu o stanie:
200 — serwer udzielił odpowiedzi i wszystko jest w porządku,
304 — niezmodyfikowany,
404 — strona nie została znaleziona,
500 — wewnętrzny błąd serwera.

Jeżeli uruchamiasz kod lokalnie, to nie otrzymasz właściwości stanu serwera. Polecenie sprawdzenia trzeba więc umieścić w komentarzu, a wartością warunku powinno być `true`. Jeżeli serwer nie zwraca właściwości `status`, sprawdź jego konfigurację.

Przeglądarka pozwoli na użycie tej techniki wczytywania danych HTML tylko wtedy, gdy pochodzą z tej samej domeny, do której należy pozostała część strony.

Niezależnie od formatu danych zwróconych przez serwer (HTML, XML lub JSON) proces konfiguracji żądania Ajax i sprawdzenia, czy plik jest gotowy do pracy, przedstawia się dokładnie tak samo. Zmianie ulega jedynie sposób pracy z otrzymanymi danymi.

W przykładzie po prawej stronie kod odpowiedzialny za wyświetlenie nowej zawartości HTML jest umieszczony wewnątrz konstrukcji warunkowej.

Uwaga: Przedstawione tutaj przykłady nie działają lokalnie w przeglądarce Chrome, ale powinny działać w przeglądarkach Firefox i Safari. W przypadku wersji IE wcześniejszych niż 9 efekty bywają różne.

W dalszej części rozdziału zobaczysz, że jQuery oferuje lepszą obsługę Ajax w różnych przeglądarkach.

1. Obiekt `XMLHttpRequest` jest przechowywany w zmiennej o nazwie `xhr`.

2. Metoda `open()` obiektu `XMLHttpRequest` przygotowuje żądanie. Metoda ta ma trzy parametry:
i). `HTTP GET` lub `POST` w celu wskazania sposobu wykonania żądania;
ii). ścieżkę dostępu do strony odpowiedzialnej za obsługę żądania;
iii). wartość boolowską, która wskazuje, czy żądanie jest asynchroniczne.

3. Do tej chwili przeglądarka jeszcze nie skontaktowała się z serwerem, aby pobrać nową zawartość HTML.

Kontakt nie nastąpi aż do wykonania ostatniego wiersza w skrypcie wywołującego metodę `send()` obiektu `XMLHttpRequest`. Metoda `send()` wymaga podania argumentu. Jeżeli nie są przekazywane żadne dane, argumentem może być `null`.

4. Po udzieleniu odpowiedzi przez serwer nastąpi wywołanie zdarzenia `onload`, co spowoduje wykonanie funkcji anonimowej.

5. Wewnątrz funkcji konstrukcja warunkowa sprawdza, czy wartością właściwości `status` obiektu jest 200, która oznacza udzielenie przez serwer prawidłowej odpowiedzi. Jeżeli przykład jest uruchamiany lokalnie, nie będzie odpowiedzi i nie można przeprowadzić sprawdzenia.

JAVASCRIPT — c08/js/data-html.js

```
1  var xhr = new XMLHttpRequest();                    // Utworzenie obiektu XMLHttpRequest.

4  xhr.onload = function() {                          // Po wczytaniu odpowiedzi.
     // Poniższa konstrukcja warunkowa nie działa lokalnie; działa jedynie w serwerze.
5    if(xhr.status === 200) {                         // Jeżeli stan serwera wskazuje,
                                                      // że wszystko jest w porządku.
6      document.getElementById('content').innerHTML = xhr.responseText;
       // Uaktualnienie.
     }
   };

2  xhr.open('GET', 'data/data.html', true);           // Przygotowanie żądania.
3  xhr.send(null);                                    // Wykonanie żądania.
```

6. Następuje uaktualnienie strony:

```
document.getElementById('content').innerHTML = xhr.responseText;.
          A                                B              C
```

A. Wybrany zostanie element przechowujący nową zawartość HTML. (W omawianym przykładzie to element, którego atrybut `id` ma wartość `content`).

B. Właściwość `innerHTML` zastępuje zawartość wskazanego elementu nową zawartością HTML pobraną z serwera.

C. Nowa zawartość HTML jest pobierana z właściwości `responseText` obiektu `XMLHttpRequest`.

Pamiętaj, że właściwość `innerHTML` powinna być używana tylko wtedy, gdy wiadomo, że serwer nie zwróci zawartości o złośliwym działaniu. Wszelka zawartość utworzona przez użytkownika lub firmy trzecie powinna być zneutralizowana w serwerze (patrz rozdział 5., podrozdział „Ataki typu XSS").

WCZYTYWANIE XML Z WYKORZYSTANIEM TECHNOLOGII AJAX

Żądanie danych XML odbywa się podobnie jak w przypadku danych HTML. Jednak przetworzenie danych zwróconych przez serwer jest znacznie bardziej skomplikowane, ponieważ dane XML muszą być skonwertowane na HTML, aby można je wyświetlić na stronie.

Na stronie po prawej możesz zobaczyć, że kod żądania pliku XML jest praktycznie identyczny z kodem żądania danych HTML pokazanym na poprzedniej stronie. Zmianie uległ fragment *wewnątrz* konstrukcji warunkowej przetwarzającej odpowiedź (punkty 1 – 4 na stronie po prawej). Dane XML muszą być skonwertowane na HTML. Struktura HTML poszczególnych wydarzeń została przedstawiona poniżej.

1. Kiedy serwer udziela odpowiedzi, przekazując dane XML, można je uzyskać za pomocą właściwości `responseXML` obiektu `XMLHttpRequest`. W omawianym przykładzie zwrócone dane XML są przechowywane w zmiennej o nazwie `response`.

2. Następnie mamy deklarację nowej zmiennej o nazwie `events`, przechowującej wszystkie elementy `<event>` z dokumentu XML. (Plik XML widziałeś w podrozdziale „XML — Extensible Markup Language").

3. Plik XML jest przetwarzany za pomocą metod DOM omówionych w rozdziale 5. Na początku z wykorzystaniem pętli `for` przeprowadzana jest iteracja przez wszystkie elementy `<event>` i następuje zebranie danych przechowywanych w ich elementach potomnych oraz umieszczenie tych danych w nowych elementach HTML.

Następnie każdy z nowych elementów HTML jest umieszczany na stronie.

4. Wewnątrz pętli `for` widzimy wielokrotne wywołanie funkcji `getNodeValue()`. Celem tych wywołań jest pobranie zawartości poszczególnych elementów XML. Wymieniona funkcja pobiera dwa parametry:
i). `obj` to fragment XML;
ii). `tag` to nazwa znacznika, z którego mają być pobrane informacje.

Funkcja wyszukuje dopasowany znacznik we fragmencie XML, używając do tego metody DOM o nazwie `getElementsByTagName()`. Następnie pobiera tekst z pierwszego elementu dopasowanego w danym fragmencie.

Dane XML każdego zdarzenia są przekształcane na następującą strukturę HTML:

HTML

```
<div class="event">
  <img src="plik.png" alt="Lokalizacja" />
  <p><b>Lokalizacja</b><br />Data wydarzenia</p>
</div>
```

JAVASCRIPT c07/js/position.js

```
var xhr = new XMLHttpRequest();                    // Utworzenie obiektu XMLHttpRequest.

xhr.onload = function() {                          // Po wczytaniu odpowiedzi.
 // Poniższa konstrukcja warunkowa nie działa lokalnie; działa jedynie w serwerze.
 if (xhr.status === 200) {
 // Jeżeli stan serwera wskazuje, że wszystko jest w porządku.

// Ten fragment jest inny, ponieważ przetwarzane są dane XML, a nie HTML.
1 var response = xhr.responseXML;                  // Pobranie danych XML z serwera.
2 var events = response.getElementsByTagName('event');
// Wyszukanie elementów <event>.

for (var i = 0; i < events.length; i++) {          // Iteracja przez znalezione elementy.
 var container, image, location, city, newline; // Deklaracja zmiennych.
 container = document.createElement('div');     // Utworzenie pojemnika <div>.
 container.className = 'event';                 // Dodanie atrybutu class.

 image = document.createElement('img');         // Dodanie obrazu mapy.
 image.setAttribute('src', getNodeValue(events[i], 'map'));
 image.appendChild(document.createTextNode(getNodeValue(events[i], 'map')));
 container.appendChild(image);

 location = document.createElement('p');
3 // Dodanie danych dotyczących lokalizacji.
 city = document.createElement('b');
 newline = document.createElement('br');
 city.appendChild(document.createTextNode(getNodeValue(events[i], 'location')));
 location.appendChild(newline);
 location.insertBefore(city, newline);
 location.appendChild(document.createTextNode(getNodeValue(events[i], 'date')));
 container.appendChild(location);

 document.getElementById('content').appendChild(container);
}
function getNodeValue(obj, tag) {                  // Pobranie zawartości z danych XML.
4 return obj.getElementsByTagName(tag)[0].firstChild.nodeValue;
}

 // Ostatni fragment jest taki sam jak w przypadku danych HTML,
 // ale żądanie dotyczy pliku XML.
 }
};
xhr.open('GET', 'data/data.xml', true);            // Przygotowanie żądania.
xhr.send(null);                                    // Wykonanie żądania.
```

WCZYTYWANIE DANYCH JSON Z WYKORZYSTANIEM TECHNOLOGII AJAX

Żądanie danych JSON opiera się na tej samej składni, którą poznałeś w przykładach dotyczących żądań danych HTML i XML. Kiedy serwer udziela odpowiedzi, dane JSON są konwertowane na postać HTML.

Kiedy dane w formacie JSON są przekazywane z serwera WWW do przeglądarki internetowej, podczas transmisji mają postać ciągu tekstowego.

Gdy ciąg tekstowy dotrze do przeglądarki, skrypt musi skonwertować go na obiekt JavaScript. Proces ten nosi nazwę **deserializacji** obiektu.

Odbywa się to z wykorzystaniem metody `parse()` wbudowanego obiektu o nazwie `JSON`. To jest obiekt globalny, a więc można go używać bez konieczności wcześniejszego utworzenia egzemplarza obiektu.

Gdy ciąg tekstowy zostanie przetworzony, skrypt może uzyskać dostęp do danych obiektu, a następnie utworzyć zawartość HTML, która będzie wyświetlona na stronie.

Zawartość HTML jest dodawana na stronie za pomocą właściwości `innerHTML`. Dlatego też powinieneś używać jej tylko wtedy, gdy masz absolutną pewność, że dodawana treść nie zawiera żadnego kodu o złośliwym działaniu (patrz ataki typu XSS w rozdziale 5., w podrozdziale „Ataki typu XSS").

Gdy przykładowy kod zostanie wykonany w przeglądarce, efekt będzie dokładnie taki sam jak w przypadku dwóch poprzednich.

Obiekt `JSON` ma również metodę o nazwie `stringify()`, która konwertuje obiekty na postać ciągu tekstowego, używając notacji JSON. To pozwala na przesyłanie obiektu z przeglądarki do serwera WWW. Proces ten nazywa się **serializacją** obiektu.

Wymienioną metodę można stosować, gdy użytkownik wykorzystuje stronę w taki sposób, że następuje uaktualnienie danych przechowywanych w obiekcie JavaScript (na przykład na skutek wypełnienia formularza sieciowego). Pozwala to na uaktualnienie informacji przechowywanych w serwerze.

Poniżej przedstawiono dane JSON, które będą przetwarzane (dane te wprowadzono w podrozdziale „Praca z danymi JSON"). Zwróć uwagę na to, że plik został zapisany wraz z rozszerzeniem *.json* w nazwie.

c08/data/data.json — JAVASCRIPT

```
{
  "events": [
    { "location": "San Francisco, CA", "date": "1 maja", "map": "img/map-ca.png" },
    { "location": "Austin, TX", "date": "15 maja", "map": "img/map-tx.png" },
    { "location": "Nowy Jork, NY", "date": "30 maja", "map": "img/map-ny.png"}
  ]
}
```

1. Dane JSON otrzymane z serwera są przechowywane w zmiennej o nazwie `responseObject`. Będą dostępne za pomocą właściwości `responseText` obiektu `XMLHttpRequest`.

Dane JSON otrzymane z serwera mają postać ciągu tekstowego. Dlatego też należy je skonwertować na obiekt JavaScript, używając metody `parse()` obiektu JSON.

2. Utworzenie zmiennej `newContent` przeznaczonej do przechowywania nowych danych HTML. Poza pętlą zmiennej jest przypisywany pusty ciąg tekstowy, a więc kod w pętli może dodawać do niego dane.

3. Iteracja przez obiekty przedstawiające poszczególne zdarzenia. Dane we wszystkich obiektach są dostępne w notacji z użyciem kropki, podobnie jak obiekty.

Wewnątrz pętli zawartość obiektu zostaje dodana do zmiennej `newContent` wraz z odpowiednim kodem znaczników HTML.

4. Gdy kończy się działanie pętli prowadzącej iterację przez obiekt `event`, w `responseObject` nowa zawartość HTML zostaje umieszczona na stronie za pomocą właściwości `innerHTML`.

JAVASCRIPT c08/js/data-json.js

```
var xhr = new XMLHttpRequest();                    // Utworzenie obiektu XMLHttpRequest.

xhr.onload = function() {                          // Po zmianie stanu.
 if(xhr.status === 200) {                          // Jeżeli stan serwera wskazuje,
                                                   // że wszystko jest w porządku.
(1)  responseObject = JSON.parse(xhr.responseText);

  // Utworzenie ciągu tekstowego wraz z nową zawartością (można użyć także operacji
  // opartych na modelu DOM).
(2)  var newContent = '';
  for (var i = 0; i < responseObject.events.length; i++) { // Iteracja przez obiekt.
    newContent += '<div class="event">';
    newContent += '<img src="' + responseObject.events[i].map + '" ';
(3)  newContent += 'alt="' + responseObject.events[i].location + '" />';
    newContent += '<p><b>' + responseObject.events[i].location + '</b><br>';
    newContent += responseObject.events[i].date + '</p>';
    newContent += '</div>';
  }

  // Uaktualnienie strony nową zawartością.
(4)  document.getElementById('content').innerHTML = newContent;

 }
};

xhr.open('GET', 'data/data.json', true);           // Przygotowanie żądania.
xhr.send(null);                                    // Wykonanie żądania.
```

PRACA Z DANYMI POCHODZĄCYMI Z INNYCH SERWERÓW

Ajax działa doskonale w przypadku danych pochodzących z Twojego serwera, ale ze względów bezpieczeństwa przeglądarki internetowe nie wczytują odpowiedzi Ajax z innych domen (odpowiedzi będących wynikiem tak zwanych żądań typu cross-domain). Dostępne są trzy najczęściej stosowane rozwiązania problemu.

PLIK PROXY W SERWERZE WWW

Pierwszy sposób na wczytanie danych z zewnętrznego serwera polega na utworzeniu we *własnym* serwerze pliku przeznaczonego na dane pochodzące ze zdalnego serwera (z wykorzystaniem języka działającego po stronie serwera, takiego jak ASP.NET, PHP, Node.js, Ruby). Strony w Twojej witrynie będą żądały danych ze wspomnianego pliku w serwerze, który z kolei pobierze odpowiednie dane z serwera zdalnego. Takie rozwiązanie nosi nazwę **proxy**, ponieważ działa w imieniu innej strony.

Rozwiązanie opiera się na tworzeniu stron w języku działającym po stronie serwera, a więc wykracza poza zakres tematyczny niniejszej książki.

JSONP

JSONP (czasem można spotkać zapis JSON-P) obejmuje dodanie na stronie elementu `<script>` odpowiedzialnego za wczytanie danych JSON z innego serwera. Takie rozwiązanie działa, ponieważ nie istnieją ograniczenia dotyczące źródła pochodzenia skryptu w elemencie `<script>`.

Skrypt zawiera wywołanie funkcji, a dane w formacie JSON są dostarczane jako argument funkcji. Wywoływana funkcja jest zdefiniowana na stronie żądającej danych i używana jest do ich przetworzenia oraz wyświetlenia. Patrz opis na kolejnej stronie.

CROSS-ORIGIN RESOURCE SHARING

W trakcie komunikacji przeglądarki internetowej i serwera WWW informacje między nimi są przekazywane za pomocą nagłówków HTTP. CORS (ang. *Cross-Origin Resource Sharing*) oznacza umieszczenie w nagłówkach HTTP informacji dodatkowych, które wskazują przeglądarce i serwerowi WWW sposób prowadzenia komunikacji.

CORS to specyfikacja W3C, ale obsługiwana jedynie w najnowszych wersjach przeglądarek. Ponieważ wymaga konfiguracji nagłówków HTTP w serwerze WWW, wykracza to poza zakres tematyczny niniejszej książki.

ALTERNATYWY

Wiele osób używa jQuery w celu wykonywania żądań pobierających dane ze zdalnych serwerów. Upraszcza to proces i zapewnia wsteczną zgodność ze starszymi przeglądarkami. Jak możesz zobaczyć w następnej kolumnie, obsługa niektórych nowych podejść może być problematyczna.

OBSŁUGA CORS

Standardowo obsługę CORS oferują przeglądarki Chrome 4, Firefox 3.5, IE10, Safari 4, Android 2.1 i iOS 3.2.

Przeglądarki Internet Explorer 8 – 9 używają niestandardowego obiektu `XDomainRequest` do obsługi żądań CORS.

JAK DZIAŁA JSONP?

Strona musi przede wszystkim zawierać funkcję przeznaczoną do przetwarzania danych w formacie JSON. Następnie za pomocą elementu `<script>` wykonywane jest żądanie danych z serwera.

Serwer zwraca plik, który wywołuje funkcję przetwarzającą dane. Dane w formacie JSON są dostarczane jako argument tej funkcji.

PRZEGLĄDARKA INTERNETOWA

Strona HTML używa dwóch fragmentów kodu JavaScript.

1. Funkcja przetwarzająca dane JSON wysłane przez serwer. W przykładzie przedstawionym na następnej stronie funkcja ta ma nazwę `showEvents()`.

2. Element `<script>`, którego atrybut `src` wskazuje dane JSON ze zdalnego serwera.

```
<script>
function showEvents(data) {
  // Kod odpowiedzialny za przetworzenie
  // danych i wyświetlenie ich na stronie.
}
</script>

<script src="http://example.org/jsonp">
</script>
```

SERWER

Kiedy serwer udziela odpowiedzi, skrypt zawiera wywołanie do nazwanej funkcji, która będzie przetwarzać dane (funkcja ta została zdefiniowana w kroku 1.). Wywołanie tej funkcji jest oznaczone literą P w skrócie JSONP. Dane w formacie JSON są dostarczane jako argument tej funkcji.

W omawianym przykładzie dane JSON znajdują się wewnątrz wywołania funkcji `showEvents()`:

```
showEvents({
  "events": [
    {
      "location": "San Francisco, CA",
      "date": "1 maja",
      "map": "img/map-ca.png"
    }...
  ]
});
```

Trzeba zwrócić uwagę na brak konieczności użycia metod `parse()` lub `stringfy()` obiektu JSON podczas pracy z JSONP. Ponieważ dane są przekazywane jako plik skryptu (a nie ciąg tekstowy), będą traktowane jako obiekt.

Plik w serwerze jest często zapisywany, a więc można podać nazwę funkcji, która będzie przetwarzała otrzymywane dane. Nazwa tej funkcji jest zwykle podawana w ciągu tekstowym zapytania adresu URL:

```
http://example.org/upcomingEvents.php?callback=showEvents
```

UŻYCIE JSONP

Przykład ten wygląda tak samo jak przykład wykorzystujący dane JSON, ale informacje szczegółowe dotyczące zdarzenia są pobierane ze zdalnego serwera. Dlatego też w kodzie HTML znajdują się dwa elementy `<script>`.

Pierwszy z nich jest odpowiedzialny za wczytanie pliku JavaScript zawierającego funkcję `showEvents()`. Wymieniona funkcja służy do wyświetlenia informacji o wydarzeniach.

Natomiast drugi element `<script>` wczytuje informacje ze zdalnego serwera. Nazwa funkcji przetwarzającej dane jest podawana w ciągu tekstowym zapytania.

c08/data-jsonp.html — HTML

```
    <script src="js/data-jsonp.js"></script>
    <script src="http://deciphered.com/js/jsonp.js?callback=showEvents"></script>
  </body>
</html>
```

c08/js/data-jsonp.js — JAVASCRIPT

```
function showEvents(data) {        // Funkcja wywołania zwrotnego po wczytaniu JSON.
  var newContent = '';             // Zmienna przechowująca zawartość HTML.

  // Utworzenie ciągu tekstowego wraz z nową zawartością (można użyć także operacji
  // opartych na modelu DOM).
  for (var i = 0; i < data.events.length; i++) {    // Iteracja przez dane.
    newContent += '<div class="event">';
    newContent += '<img src="' + data.events[i].map + '"';
    newContent += ' alt="' + data.events[i].location + '" />';
    newContent += '<p><b>' + data.events[i].location + '</b><br>';
    newContent += data.events[i].date + '</p>';
    newContent += '</div>';
  }

  // Uaktualnienie strony nową zawartością.
  document.getElementById('content').innerHTML = newContent; }
```

1. Kod w pętli `for` (wykorzystywanej do przetworzenia danych JSON oraz utworzenia zawartości HTML) i wiersz kodu odpowiedzialny za umieszczenie zawartości na stronie są takie same jak w przykładzie przetwarzającym dane JSON pobrane z tego samego serwera.

Istnieją trzy ważne różnice.
i) Kod jest opakowany funkcją o nazwie `showEvents()`.
ii) Dane JSON są podawane jako argument wywołania funkcji.
iii) Dane nie muszą być przetwarzane za pomocą metody `JSON.parse()`. W pętli `for` odwołanie do danych następuje z zastosowaniem parametru `data`.

Zamiast definiować drugi element `<script>` w dokumentach HTML, można wykorzystać JavaScript do wstawienia nowego elementu `<script>` na stronie HTML (podobnie jak dodajesz na niej każdy inny element). Dzięki temu cała funkcjonalność dotycząca zewnętrznych danych będzie znajdowała się w jednym pliku JavaScript.

JSONP wczytuje JavaScript, a każde dane JavaScript mogą zawierać kod o złośliwym działaniu. Z tego powodu powinieneś wczytywać dane jedynie z zaufanych źródeł.

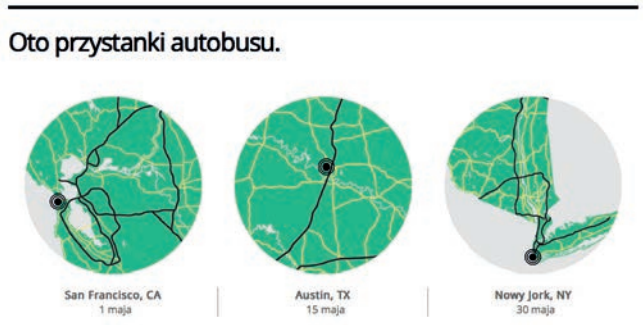
Ponieważ JSONP oznacza wczytywanie danych z innego serwera, możesz dodać zegar w celu sprawdzenia, czy serwer udzielił odpowiedzi w określonym czasie (jeśli odpowiedź nie nadejdzie, można wyświetlić komunikat o błędzie).

Więcej informacji dotyczących obsługi błędów znajdziesz w rozdziale 10., natomiast przykład użycia zegara przedstawiono w rozdziale 11. (w którym tworzymy slajdy z zawartością).

JAVASCRIPT http://htmlandcssbook.com/js/jsonp.js

```
showEvents({
  "events": [
     {
       "location": "San Francisco, CA",
       "date": "1 maja",
       "map": "img/map-ca.png"
     },
     {
       "location": "Austin, TX",
       "date": "15 maja",
       "map": "img/map-tx.png"
     },
     {
       "location": "New York, NY",
       "date": "30 maja",
       "map": "img/map-ny.png"
     }
   ]
});
```

WYNIK



Zawartość pliku zwróconego z serwera jest w wywołaniu funkcji `showEvents()` opakowana danymi w formacie JSON. Dlatego też funkcja `showEvents()` będzie wywoływana tylko wtedy, gdy przeglądarka internetowa wczyta zewnętrzne dane.

JQUERY I AJAX — ŻĄDANIA

jQuery oferuje kilka metod przeznaczonych do obsługi żądań Ajax. Podobnie jak inne przykłady w rozdziale, proces ten składa się z dwóch kroków — wykonania żądania i obsługi otrzymanej odpowiedzi.

W tabeli po prawej stronie wymieniono sześć metod jQuery pozwalających na wykonywanie żądań Ajax. Pierwsze pięć to skróty metody `$.ajax()`, która została wymieniona jako ostatnia (szósta).

Metoda `.load()` operuje na elementach wybranych w jQuery (czyli działa podobnie jak większość metod jQuery). Nową zawartość HTML metoda ta umieszcza we wskazanych elementach.

Możesz zobaczyć, że zapis pozostałych pięciu metod jest zupełnie inny. To są metody obiektu globalnego `jQuery`, stąd ich nazwy zaczynają się od znaku $. Metody te jedynie żądają danych z serwera; nie używają automatycznie tych danych w celu uaktualnienia elementów znajdujących się w dopasowanym zbiorze. Dlatego też po znaku $ nie znajduje się selektor.

Kiedy serwer zwróci dane, skrypt musi wskazać, w jaki sposób mają być przetworzone.

METODA (SKŁADNIA)	OPIS
`.load()`	Wczytuje fragmenty HTML w elemencie. To jest najprostsza metoda przeznaczona do pobierania danych.
`$.get()`	Wczytuje dane za pomocą metody HTTP `GET`. Używana do **żądania** danych z serwera.
`$.post()`	Wczytuje dane za pomocą metody HTTP `POST`. Używana w celu **wysyłania** danych, które uaktualniają dane w serwerze.
`$.getJSON()`	Wczytuje dane JSON za pomocą żądania `GET`. Metoda używana dla danych JSON.
`$.getScript()`	Wczytuje i wykonuje dane JavaScript za pomocą żądania `GET`. Metoda używana dla danych JavaScript (na przykład JSONP).
`$.ajax()`	Metoda wykorzystywana do wykonywania wszystkich żądań. Wymienione wcześniej metody w tle używają `$.ajax()`.

JQUERY I AJAX — ODPOWIEDZI

Podczas użycia metody `.load()` zawartość HTML zwracana przez serwer zostaje wstawiona do elementów wybranych w jQuery. W przypadku pozostałych metod trzeba wskazać sposób przetworzenia danych zwróconych za pomocą obiektu `jqXHR`.

WŁAŚCIWOŚCI JQXHR	OPIS
`responseText`	Zwrócone dane tekstowe.
`responseXML`	Zwrócone dane XML.
`status`	Kod stanu.
`statusText`	Opis stanu (zwykle wykorzystywany do wyświetlenia informacji o błędzie, jeśli taki wystąpi).

METODY JQXHR	OPIS
`.done()`	Kod przeznaczony do uruchomienia, jeśli wykonanie żądania zakończyło się powodzeniem.
`.fail()`	Kod przeznaczony do uruchomienia, jeśli wykonanie żądania zakończyło się niepowodzeniem.
`.always()`	Kod przeznaczony do uruchomienia, jeśli wykonanie żądania zakończyło się powodzeniem lub niepowodzeniem.
`.abort()`	Wstrzymanie komunikacji.

jQuery ma obiekt o nazwie `jqXHR`, który ułatwia obsługę danych otrzymanych z serwera. Właściwości i metody tego obiektu (wymienione w tabelach po lewej stronie) będą używane na kilku kolejnych stronach.

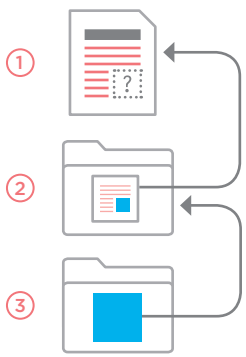
Ponieważ jQuery pozwala na łączenie metod, to za pomocą metod `.done()`, `.fail()` i `.always()` można uruchomić odpowiedni kod w zależności od wyniku operacji wczytywania danych.

WZGLĘDNE ADRESY URL

Jeżeli zawartość wczytywana za pomocą technologii Ajax zawiera względne adresy URL (na przykład obrazy i łącza), to te adresy URL będą traktowane w taki sposób, jakby były względne dla *pierwotnie* wczytanej strony.

Jeżeli nowa zawartość HTML znajduje się w innym katalogu niż pierwotna strona, to względne ścieżki adresu mogą okazać się nieprawidłowe.

1. Ten plik HTML wykorzystuje technologię Ajax w celu wczytania zawartości ze strony znajdującej się w katalogu wskazanym w punkcie 2.
2. Strona w tym katalogu zawiera obrazy o ścieżkach dostępu względnych dla drugiego katalogu: `<img src="img/box.gif" />`.
3. Plik HTML nie może odszukać obrazu, ponieważ ścieżka dostępu nie jest już prawidłowa — nie prowadzi do elementu w katalogu potomnym.



WCZYTYWANIE ZAWARTOŚCI HTML NA STRONIE Z WYKORZYSTANIEM JQUERY

Metoda `.load()` to najprostsza z metod jQuery Ajax. Może być używana jedynie do wczytywania zawartości HTML z serwera, ale kiedy serwer udziela odpowiedzi, zawartość ta jest umieszczana w elementach wybranych w jQuery.

SELEKTOR JQUERY

Na początku należy wybrać element, w którym ma zostać umieszczona zawartość HTML.

ADRES URL STRONY

Następnie metoda `.load()` jest używana do wskazania adresu URL strony HTML przeznaczonej do wczytania.

SELEKTOR

Istnieje możliwość określenia, że ma zostać wczytana tylko część strony (zamiast całej).

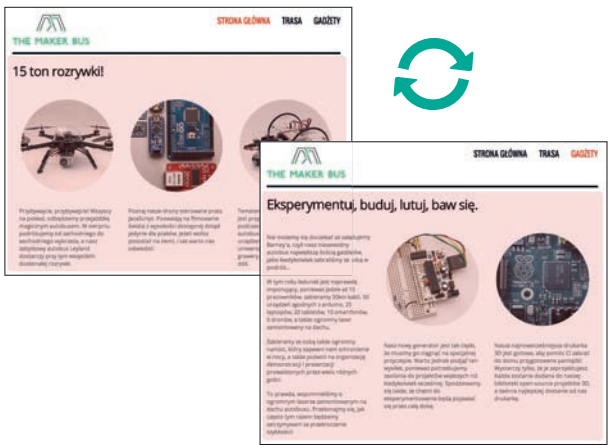
```
$('#content').load('jq-ajax3.html #content');
```

1. Utworzenie obiektu jQuery, którego atrybut `id` ma wartość `content`.

2. Adres URL strony, z której ma zostać wczytana zawartość HTML. Między adresem URL i selektorem w kroku 3. musi znajdować się spacja.

3. To jest fragment strony HTML przeznaczony do wyświetlenia. Ponownie będzie to sekcja, której atrybut `id` ma wartość `content`.

Na stronie pokazanej po prawej łącza znajdujące się w prawym górnym rogu są używane do przejścia na inne strony. Jeżeli użytkownik ma włączoną obsługę JavaScript, to kliknięcie łącza powoduje, że kod w metodzie obsługi zdarzenia `.on()` nie wczytuje całej nowej strony. Zamiast tego metoda `.load()` zastępuje obszar oznaczony kolorem różowym (sekcja, w której atrybut `id` ma wartość `content`) odpowiednim fragmentem ze strony wskazanej przez użytkownika. Odświeżony będzie jedynie obszar oznaczony kolorem różowym, a nie cała strona.



WCZYTYWANIE ZAWARTOŚCI

Kiedy użytkownik kliknie dowolne łącze w elemencie `<nav>`, nastąpi jedno z dwóch poniższych zdarzeń.

Jeżeli włączona jest obsługa JavaScript, zdarzenie `click` spowoduje wywołanie funkcji anonimowej, która z kolei wczyta i umieści na stronie nową zawartość.

Jeżeli obsługa JavaScript nie jest włączona, to nastąpi standardowe przejście z jednej strony na inną.

Wewnątrz funkcji anonimowej można wyróżnić pięć etapów działania.

1. Wywołanie `e.preventDefault()` wstrzymuje przeniesienie użytkownika na nową stronę.

2. Zmienna o nazwie `url` przechowuje adres URL strony do wczytania. Adres ten jest pobierany z atrybutu `href` łącza kliknięte przez użytkownika i wskazuje stronę przeznaczoną do wczytania.

3. Atrybuty `class` w łączach zostają uaktualnione i wskazują stronę bieżącą.

4. Element przechowujący zawartość zostaje usunięty.

5. Następuje wybór elementu kontenera, a metoda `.load()` pobiera nową zawartość. Element jest natychmiast ukrywany za pomocą wywołania `.hide()`, aby mógł pojawić się na stronie na skutek wywołania `fadeIn()`.

JAVASCRIPT c08/js/jq-load.js

```
$('nav a').on('click', function(e) {          // Użytkownik kliknął łącze.
  e.preventDefault();                         // Zatrzymanie wczytywania nowego łącza.
  var url = this.href;                        // Pobranie wartości atrybutu href.

  $('nav a.current').removeClass('current');  // Usunięcie klasy current.
  $(this).addClass('current');    // Określenie nowego elementu jako bieżącego.

  $('#container').remove();       // Usunięcie starej zawartości.
  $('#content').load(url + ' #content').hide().fadeIn('slow');  // Nowa zawartość.
});
```

HTML c08/jq-load.html

```
<nav>
  <a href="jq-load.html" class="current">Strona główna</a>
  <a href="jq-load2.html">Trasa</a>
  <a href="jq-load3.html">Gadżety</a>
</nav>
<section id="content">
  <div id="container">
    <!-- Miejsce na zawartość strony. -->
  </div>
</section>
```

Łącza będą działały, nawet jeśli obsługa JavaScript jest wyłączona w przeglądarce. W przypadku włączenia obsługi JavaScript biblioteka jQuery zawartość wskazanej strony wczyta do elementu `<div>`, którego atrybut `id` ma wartość `content`. Pozostała część strony nie musi być ponownie wczytywana.

SKRÓTY METOD AJAX W JQUERY

jQuery oferuje skróty metod przeznaczonych do obsługi określonych typów żądań Ajax.

Wymienione poniżej metody są metodami skrótów. Patrząc na kod źródłowy jQuery, zauważysz, że w tle wszystkie one używają metody `$.ajax()`.

Metody te poznasz na kolejnych kilku stronach; wprowadzają one kluczowe aspekty metody `$.ajax()`.

W przeciwieństwie do innych metod wymienione tutaj nie działają z elementami wybranymi w jQuery. Dlatego też jedynym prefiksem jest znak $ zamiast kolekcji elementów wybranych w jQuery. Z reguły metody te są wywoływane przez zdarzenie, takie jak wczytanie strony lub działania podejmowane przez użytkownika na stronie (na przykład kliknięcie łącza, wysłanie formularza).

W przypadku żądania Ajax często zachodzi potrzeba wysłania danych do serwera, co z kolei wpływa na dane przekazywane przeglądarce przez serwer.

Podobnie jak w przypadku formularzy HTML (i żądań Ajax przedstawionych wcześniej w rozdziale), dane można wysłać za pomocą HTTP `GET` lub `POST`.

METODA (SKŁADNIA)	OPIS
`$.get(`*`url`*`[, `*`dane`*`][, `*`wywołanie_zwrotne`*`][, `*`typ`*`])`	Żądanie HTTP `GET` mające na celu pobranie danych.
`$.post(`*`url`*`[, `*`dane`*`][, `*`wywołanie_zwrotne`*`][, `*`typ`*`])`	Żądanie HTTP `POST` uaktualniające dane w serwerze.
`$.getJSON(`*`url`*`[, `*`dane`*`][, `*`wywołanie_zwrotne`*`])`	Wczytanie danych JSON za pomocą żądania `GET`.
`$.getScript(`*`url`*`[, `*`wywołanie_zwrotne`*`])`	Wczytanie i wykonanie kodu JavaScript (na przykład JSONP) za pomocą żądania `GET`.

Parametry wymienione w nawiasach kwadratowych są opcjonalne.

`$` wskazuje, że jest to metoda obiektu `jQuery`.
`url` określa miejsce, skąd mają być pobrane dane.
`dane` dostarcza wszelkie informacje dodatkowe przekazywane serwerowi.
`wywołanie_zwrotne` wskazuje, że funkcja powinna być wywołana, gdy zostaną zwrócone dane (może to być funkcja anonimowa lub nazwana).
`typ` wskazuje typ danych oczekiwanych z serwera.

Uwaga: Przedstawione tutaj przykłady działają jedynie w serwerze WWW (a nie lokalnym systemie plików). Konfiguracja serwera WWW i języki działające po stronie serwera to zagadnienia wykraczające poza zakres tematyczny tej książki. Przykłady możesz jednak wypróbować w witrynie poświęconej książce. Pliki PHP znajdują się w materiałach dołączonych do książki, ale umieszczono je tam tylko w celach demonstracyjnych.

ŻĄDANIE DANYCH

W przedstawionym poniżej przykładzie użytkownik głosuje na ulubioną koszulkę i nie musi przy tym opuszczać strony.

1. Kliknięcie koszulki przez użytkownika powoduje wywołanie funkcji anonimowej.

2. Metoda `e.PreventDefault()` uniemożliwia przejście na nową stronę.

3. Wybór dokonany przez użytkownika to wartość atrybutu `id` obrazu. Wartość ta jest przechowywana w zmiennej o nazwie `queryString` w formacie ciągu tekstowego zapytania, na przykład `vote=gray`.

4. Metoda `$.get()` jest wywoływana z użyciem trzech parametrów:
i) Strona odpowiedzialna za przetworzenie żądania (w tym samym serwerze).
ii) Dane wysyłane do serwera (w omawianym przykładzie jest to ciąg tekstowy zamówienia, ale to mogą być również dane JSON).
iii) Funkcja obsługująca dane otrzymane z serwera. W omawianym przykładzie to funkcja anonimowa.

5. Po udzieleniu odpowiedzi przez serwer wywołanie zwrotne w postaci funkcji anonimowej zajmuje się obsługą danych. W omawianym przykładzie kod funkcji powoduje pobranie elementu przedstawiającego koszulkę wybraną przez użytkownika, a następnie zastąpienie go kodem HTML otrzymanym z serwera. Odbywa się to za pomocą metody jQuery o nazwie `.html()`.

JAVASCRIPT c08/js/jq-get.js

```
1 $('#selector a').on('click', function(e) {
2  e.preventDefault();
3    var queryString = 'vote=' + event.target.id;
4    $.get('votes.php', queryString, function(data) {
5      $('#selector').html(data);
   });
});
```

HTML (Ten kod HTML jest tworzony przez kod znajdujący się w pliku JS).

```
<div class="third"><a href="vote.php?vote=gray">
  <img src="img/t-gray.png" id="gray" alt="gray" /></a></div>
<div class="third"><a href="vote.php?vote=yellow">
  <img src="img/t-yellow.png" id="yellow" alt="yellow" /></a></div>
<div class="third"><a href="vote.php?vote=green">
  <img src="img/t-green.png" id="green" alt="green" /></a></div>
```

WYNIK

Łącza prowadzące do koszulek są tworzone przez kod znajdujący się w pliku JS. Gwarantuje to ich wyświetlenie tylko wtedy, gdy przeglądarka obsługuje JavaScript (wynikowa struktura HTML została przedstawiona powyżej). Kiedy serwer udziela odpowiedzi, nie musi ona zawierać kodu HTML. Wartością zwrotną mogą być dowolnego rodzaju dane, które przeglądarka potrafi przetworzyć i wykorzystać.

WYSYŁANIE FORMULARZY SIECIOWYCH Z WYKORZYSTANIEM TECHNOLOGII AJAX

W celu wysłania danych do serwera prawdopodobnie użyjesz metody `.post()`. Biblioteka jQuery oferuje także metodę `.serialize()` przeznaczoną do zbierania danych z formularza sieciowego.

WYSYŁANIE DANYCH FORMULARZA SIECIOWEGO

Metoda HTTP `POST` jest często stosowana podczas wysyłania danych formularza sieciowego do serwera. Posiada odpowiadającą jej funkcję — metodę `.post()`, która pobiera trzy takie same parametry jak metoda `.get()`:

i) nazwę pliku (w tym samym serwerze) odpowiedzialnego za przetworzenie danych formularza sieciowego;
ii) dane formularza wysyłane do serwera;
iii) funkcję wywołania zwrotnego obsługującą odpowiedź udzieloną przez serwer.

Na stronie po prawej możesz zobaczyć zastosowanie metody `$.post()` wraz z `.serialize()`, co stanowi bardzo użyteczne połączenie podczas pracy z formularzami sieciowymi. Wymienione metody razem wysyłają dane do serwera.

POBIERANIE DANYCH FORMULARZA SIECIOWEGO

Działanie metody jQuery `.serialize()` przedstawia się następująco:

- pobranie wszystkich informacji z formularza sieciowego;
- umieszczenie tych informacji w ciągu tekstowym, który jest gotowy do wysłania do serwera;
- zakodowanie znaków, które nie mogą być używane w ciągu tekstowym zapytania.

Zwykle metoda ta będzie stosowana w selekcji zawierającej element `<form>` (choć można ją wykorzystywać także w poszczególnych elementach lub podzbiorze formularza).

Wysyła ona jedynie dane z kontrolek formularza sieciowego oznaczonych jako *prawidłowo wypełnione*, co oznacza, że nie zostaną wysłane informacje:

- z kontrolek wyłączonych;
- z kontrolek, w których nie wybrano żadnej opcji;
- z przycisku wysyłającego formularz.

PO STRONIE SERWERA

Kiedy strona po stronie serwera zajmuje się obsługą formularza sieciowego, można ją wykorzystać także w następujących sytuacjach:

- Wykonane zostało zwykłe żądanie strony internetowej (w takim przypadku wysyłana jest cała strona).
- Wykonane zostało żądanie Ajax (w takim przypadku wysyłany może być tylko fragment strony).

Za pośrednictwem nagłówka `X-Requested-With` po stronie serwera można sprawdzić, czy żądanie zostało wykonane w technologii Ajax.

Jeżeli wymieniony nagłówek jest ustawiony i ma wartość `XMLHttpRequest`, to wiadomo, że żądanie zostało wykonane w technologii Ajax.

WYSYŁANIE FORMULARZY SIECIOWYCH

1. Kiedy użytkownik wysyła formularz sieciowy, następuje wykonanie funkcji anonimowej.

2. Metoda `e.PreventDefault()` uniemożliwia wysłanie formularza sieciowego.

3. Za pomocą metody `.serialize()` następuje zebranie danych formularza, a następnie ich umieszczenie w zmiennej `details`.

4. Metoda `$.post()` jest wywołana z użyciem trzech parametrów:

i) adresu URL strony, do której są przekazywane dane;

ii) danych zebranych z formularza sieciowego;

iii) funkcji wywołania zwrotnego, która wyświetli wyniki użytkownikowi.

5. Kiedy serwer udzieli odpowiedzi, zawartość elementu, którego atrybut `id` ma wartość `register`, zostanie nadpisana nową zawartością HTML otrzymaną z serwera.

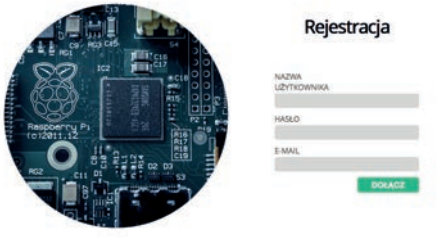
JAVASCRIPT c08/js/jq-post.js

```
1 $('#register').on('submit', function(e) {      // Po wysłaniu formularza sieciowego.
2   e.preventDefault();                          // Uniemożliwienie wysłania formularza.
3   var details = $('#register').serialize();    // Serializacja danych formularza.
4   $.post('register.php', details, function(data) {
    // Użycie metody $.post() do wysłania danych.
5     $('#register').html(data);                 // Miejsce wyświetlenia wyniku.
    });
});
```

HTML c08/jq-post.html

```
<form id="register" action="register.php" method="post">
  <h2>Rejestracja</h2>
  <label for="name">Nazwa użytkownika</label><input type="text" id="name" name="name" />
  <label for="pwd">Hasło</label><input type="password" id="pwd" name="pwd" />
  <label for="email">E-mail</label><input type="email" id="email" name="email" />
  <input type="submit" value="Dołącz" />
</form>
```

WYNIK



Przykład ten trzeba uruchomić w serwerze WWW. Strona znajdująca się po stronie serwera zwróci komunikat potwierdzenia (ale nie przeprowadza weryfikacji otrzymanych danych oraz nie wysyła wiadomości e-mail z potwierdzeniem).

WCZYTYWANIE DANYCH JSON I OBSŁUGA BŁĘDÓW AJAX

Dane JSON można wczytać za pomocą metody `$.getJSON()`. Istnieją także metody pomagające w przetworzeniu odpowiedzi, gdy wykonanie metody zakończy się niepowodzeniem.

WCZYTANIE DANYCH JSON

Jeżeli chcesz wczytać dane JSON, do dyspozycji masz metodę o nazwie `$.getJSON()`. Jej działanie polega na pobraniu danych JSON z tego samego serwera, z którego pochodzi strona. Aby użyć JSONP, należy wykorzystać metodę `$.getScript()`.

AJAX I BŁĘDY

Czasami zdarza się, że żądanie strony internetowej zakończy się niepowodzeniem; technologia Ajax nie jest tutaj wyjątkiem. Dlatego też jQuery dostarcza dwie metody, które mogą uruchomić odpowiedni kod w zależności od wyniku wykonania żądania (sukces lub niepowodzenie). Ponadto dostępna jest jeszcze trzecia metoda, uruchamiająca kod niezależnie od wyniku wykonania żądania.

Poniżej przedstawiono przykład pokazujący te koncepcje.

SUKCES I NIEPOWODZENIE

Dostępne są trzy metody, które można łączyć po `$.get()`, `$.post()`, `$.getJSON()` oraz `$.ajax()` w celu obsłużenia sukcesu lub niepowodzenia żądania:

`.done()` — metoda zdarzenia, która jest wywoływana, gdy wykonanie żądania zakończy się powodzeniem;
`.fail()` — metoda zdarzenia, która jest wywoływana, gdy wykonanie żądania zakończy się niepowodzeniem;
`.always()` — metoda zdarzenia, która jest wywoływana po wykonaniu żądania niezależnie od jego wyniku (sukces lub niepowodzenie).

W starszych skryptach można spotkać się z użyciem metod `.success()`, `.error()` i `.complete()` zamiast wymienionych. Ich działanie jest takie samo, ale nowsze metody są preferowaną opcją, począwszy od jQuery 1.8.

1. W omawianym przykładzie dane JSON przedstawiają kursy wymiany walut wczytane na stronie za pomocą funkcji o nazwie `loadRates()`.

2. W pierwszym wierszu skryptu na stronie zostaje umieszczony element przeznaczony do przechowywania danych.

3. Funkcja `loadRates()` jest wywoływana w ostatnim wierszu skryptu.

4. Wewnątrz funkcji `loadRates()` metoda `$.getJSON()` próbuje wczytać pewne dane JSON. Po wymienionej metodzie mamy dołączone wywołania jeszcze trzech innych. Nie wszystkie z nich zostaną wywołane.

5. Metoda `.done()` jest wykonana tylko wtedy, gdy pobranie danych zakończy się sukcesem. Zawiera funkcję anonimową wyświetlającą kursy wymiany i godzinę ich wyświetlenia.

6. Metoda `.fail()` jest wykonywana tylko wtedy, gdy serwer nie może zwrócić danych. Jej zadaniem jest wyświetlenie użytkownikowi komunikatu o błędzie.

7. Metoda `.always()` będzie wykonana niezależnie od wyniku żądania. Dodaje przycisk odświeżenia oraz procedurę obsługi zdarzeń ponownie wywołującą funkcję `loadRates()`.

JAVASCRIPT — c08/js/jq-getJSON.js

```
(2) $('#exchangerates').append('<div id="rates"></div><div id="reload"></div>');

(1) function loadRates() {
(4)   $.getJSON('data/rates.json')
(5)   .done( function(data){                              // Serwer zwraca dane.
        var d = new Date();                               // Utworzenie obiektu daty.
        var hrs = d.getHours();                           // Określenie godziny.
        var mins = d.getMinutes();                        // Określenie minuty.
        var msg = '<h2>Kursy wymiany walut</h2>';         // Początek komunikatu.
        $.each(data, function(key, val) {      // Dodanie poszczególnych kursów wymiany.
          msg += '<div class="' + key + '">' + key + ': ' + val + '</div>';
        });
        msg += '<br>Ostatnia aktualizacja: ' + hrs + ':' + mins + '<br>';
        // Wyświetlenie uaktualnionej godziny.
        $('#rates').html(msg);                  // Umieszczenie na stronie kursów wymiany.
(6)   }).fail( function() {                     // Wystąpił błąd.
        $('aside').append('Przepraszamy, nie można pobrać danych.');
        // Wyświetlenie komunikatu o błędzie.
(7)   }).always( function() {                   // Ta metoda zawsze jest wywoływana.
         var reload = '<a id="refresh" href="#">';
         // Dodanie łącza odświeżającego zawartość.
         reload += '<img src="img/refresh.png" alt="refresh" /></a>';
         $('#reload').html(reload);             // Dodanie łącza odświeżającego zawartość.
         $('#refresh').on('click', function(e) {
         // Dodanie procedury obsługi zdarzeń click.
           e.preventDefault();                  // Uniemożliwienie przejścia na nową stronę.
           loadRates();                         // Wywołanie funkcji loadRates().
         });
      });
    }

(3) loadRates();                                // Wywołanie funkcji loadRates().
```

WIĘKSZA KONTROLA NAD ŻĄDANIAMI AJAX

Metoda `$.ajax()` daje większą kontrolę nad żądaniami Ajax. W tle jest ona wykorzystywana w jQuery przez wszystkie metody skrótu dotyczące żądań Ajax.

W bibliotece jQuery metoda `$.ajax()` jest wykorzystywana przez inne metody pomocnicze Ajax, które zostały dotąd omówione (pozwalają one na znacznie prostsze wykonywanie żądań Ajax).

Metoda `$.ajax()` oferuje większą kontrolę nad całym procesem, udostępnia ponad 30 różnych ustawień pozwalających na kontrolę nad żądaniem Ajax. W tabeli poniżej wymieniono wybrane ustawienia. Ustawienia te są dostarczane za pomocą notacji literału obiektu (będzie to obiekt ustawień).

Przykład przedstawiony na stronie po prawej wygląda i działa podobnie jak przykład demonstrujący działanie metody `.load()` w podrozdziale „Wczytywanie zawartości HTML na stronie z wykorzystaniem jQuery". Jednak tutaj została użyta metoda `$.ajax()`.

- Ustawienia mogą pojawiać się w dowolnej kolejności, o ile wykorzystują prawidłową notację literału JavaScript.
- Ustawienia mogą pobierać funkcję, która z kolei może używać funkcji nazwanej lub anonimowej.
- Metoda `$.ajax()` nie pozwala na wczytywanie tylko fragmentu strony. Dlatego też metoda jQuery o nazwie `.find()` jest używana do wybrania interesującej Cię części strony.

USTAWIENIE	OPIS
`type`	Może pobierać wartość `GET` lub `POST` w zależności od metody, za pomocą której jest wykonywane żądanie: HTTP `GET` lub `POST`.
`url`	Strona, do której kierowane jest żądanie.
`data`	Dane wysyłane do serwera wraz z żądaniem.
`success`	Funkcja uruchamiana, gdy wykonanie żądania Ajax zakończy się powodzeniem; ustawienie podobne do metody `.done()`.
`error`	Funkcja uruchamiana, gdy wykonanie żądania Ajax zakończy się niepowodzeniem; ustawienie podobne do metody `.fail()`.
`beforeSend`	Funkcja (anonimowa lub nazwana) uruchamiana przez wykonaniem żądania Ajax. W przykładzie pokazanym na stronie po prawej służy do wyświetlenia ikony oznaczającej wczytywanie danych.
`complete`	Funkcja wykonywana po zakończeniu żądania, niezależnie od jego stanu (sukces lub niepowodzenie). W przykładzie pokazanym na stronie po prawej służy do usunięcia ikony oznaczającej wczytywanie danych.
`timeout`	Określa liczbę milisekund, które muszą upłynąć, zanim wystąpi zdarzenie oznaczające niepowodzenie.

KONTROLA TECHNOLOGII AJAX

Kiedy użytkownik kliknie łącze w elemencie <nav>, na stronie zostaje umieszczona nowa zawartość. Jest to rozwiązanie bardzo podobne do przykładu pokazanego w podrozdziale „Wczytywanie zawartości HTML na stronie z wykorzystaniem jQuery", dotyczącego metody `.load()`, ale wymagała ona tylko jednego wiersza kodu.

1. W tym przykładzie procedura obsługi zdarzeń wywołuje metodę `$.ajax()`.

Tutaj mamy siedem ustawień zastosowanych w metodzie `$.ajax()`. Pierwsze trzy to właściwości, natomiast cztery ostatnie to funkcje anonimowe wywoływane na różnych etapach żądania Ajax.

2. Przykład ten powoduje ustawienie właściwości `timeout` nakazującej dwusekundowe oczekiwanie na odpowiedź Ajax.

3. Kod umieszcza na stronie także elementy przeznaczone do wyświetlenia wczytywanych danych. Możesz ich nie zobaczyć, jeśli żądanie zostanie obsłużone szybko. Jednak na pewno je dostrzeżesz, gdy strona jest wczytywana wolno.

4. Jeżeli wykonanie żądania Ajax zakończy się niepowodzeniem, to użytkownikowi zostanie wyświetlony komunikat o błędzie.

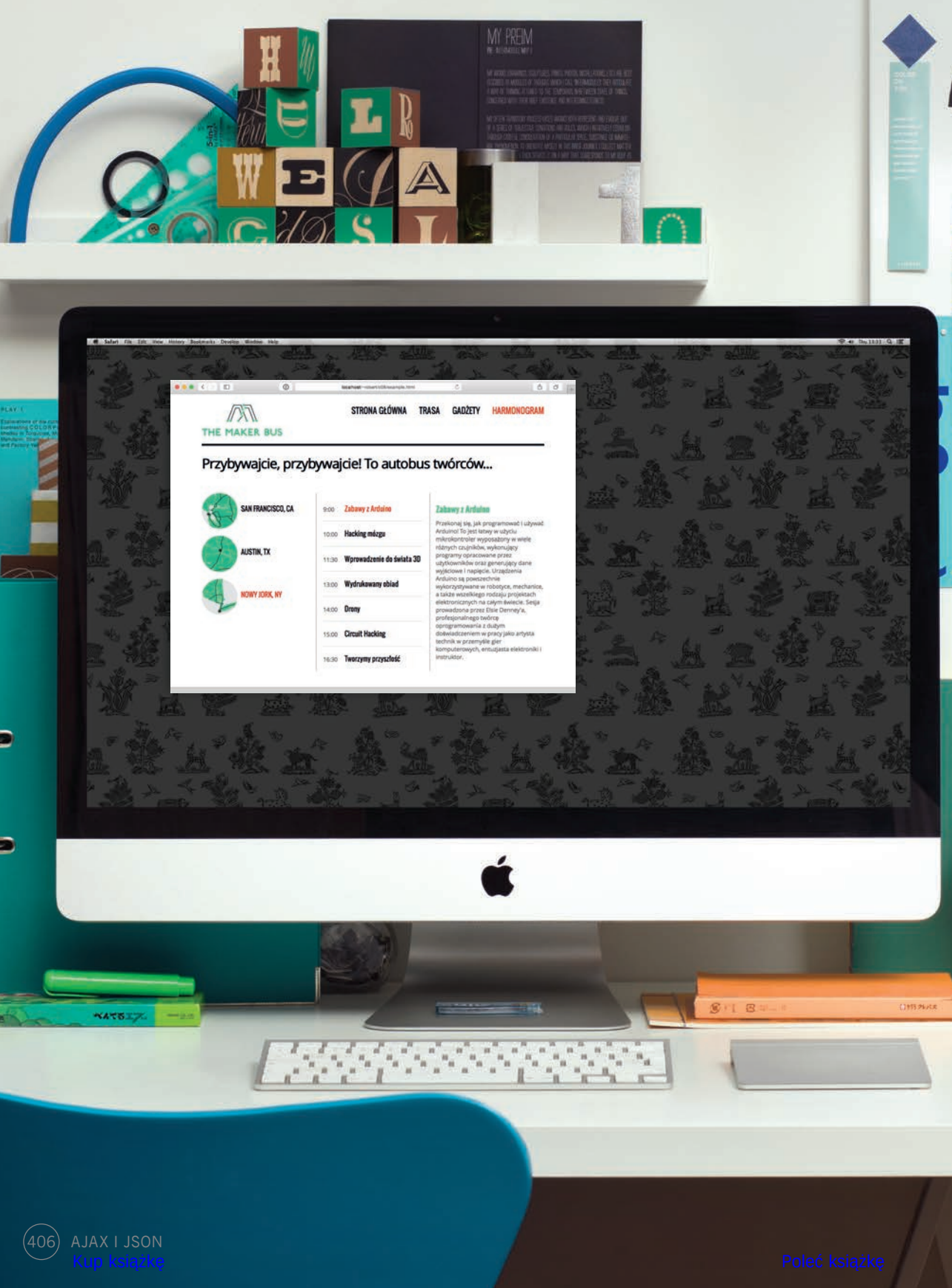
JAVASCRIPT c08/js/jq-ajax.js

```
(1) $('nav a').on('click', function(e) {
      e.preventDefault();
      var url = this.href;                          // Adres URL strony do wczytania.
      var $content = $('#content');                 // Buforowanie wyboru.

      $('nav a.current').removeClass('current');    // Uaktualnienie łączy.
      $(this).addClass('current');
      $('#container').remove();                     // Usunięcie zawartości.

      $.ajax({
        type: "POST",                               // Wybór metody: GET lub POST.
        url: url,                                   // Ścieżka dostępu do pliku.
(2)     timeout: 2000,                              // Czas oczekiwania.
        beforeSend: function() {                    // Przed wykonaniem żądania Ajax.
(3)       $content.append('<div id="load">Wczytywanie</div>');  // Wczytanie komunikatu.
        },
        complete: function() {                      // Po wykonaniu żądania Ajax.
          $('#loading').remove();                   // Usunięcie komunikatu.
        },
        success: function(data) {                   // Wyświetlenie zawartości.
          $content.html( $(data).find('#container') ).hide().fadeIn(400);
        },
        fail: function() {                          // Wyświetlenie komunikatu o błędzie.
(4)       $('#panel').html('<div class="loading">Proszę spróbować wkrótce.</div>');
        }
      });

    });
```



PRZYKŁAD
AJAX I JSON

W tym przykładzie wyświetlane są informacje dotyczące trzech wydarzeń. Dane pochodzą z trzech różnych źródeł.

1) Podczas wczytywania strony informacje dotyczące miejsc wydarzeń są umieszczane w kodzie HTML. Kiedy użytkownik kliknie zdarzenie w lewej kolumnie, środkowa będzie zawierać harmonogram wybranego zdarzenia.

W lewej kolumnie łącza mają atrybut `id`, którego wartość to dwuliterowy identyfikator stanu w USA, w którym będzie miało miejsce dane wydarzenie:

```
<a id="tx" href="tx.html">... Austin, TX</a>
```

2) Harmonogramy wydarzeń są przechowywane w obiekcie JSON, w zewnętrznym pliku pobranym podczas wczytywania modelu DOM. Gdy użytkownik kliknie wybraną sesję w środkowej kolumnie, jej opis pojawi się w kolumnie prawej.

W środkowej kolumnie wyświetlającej harmonogramy wydarzeń tytuł każdej sesji jest wstawiany wewnątrz łącza powodującego wyświetlenie informacji o danej sesji.

```
<a href="descriptions.html#Circuit-Hacking">
Circuit Hacking</a>
```

3) Opisy wszystkich sesji są przechowywane w pliku HTML. Poszczególne opisy sesji są wybierane za pomocą metody jQuery o nazwie `.load()` oraz selektora # przedstawionego w podrozdziale „Wczytywanie zawartości HTML na stronie z wykorzystaniem jQuery”.

W prawej kolumnie opis sesji jest pobierany z pliku HTML. Każda sesja jest przechowywana w elemencie, którego atrybut `id` zawiera tytuł sesji (wraz ze spacjami zastąpionymi łącznikami).

```
<div id="Wprowadzenie-do-modelowania-3D">
  <h3>Wprowadzenie do świata 3D</h3>
  <p>Przyjdź i przekonaj się, jak tworzyć modele 3D...</p>
</div>
```

Ponieważ łącza są dodawane i usuwane, w przykładzie użyto delegacji zdarzeń.

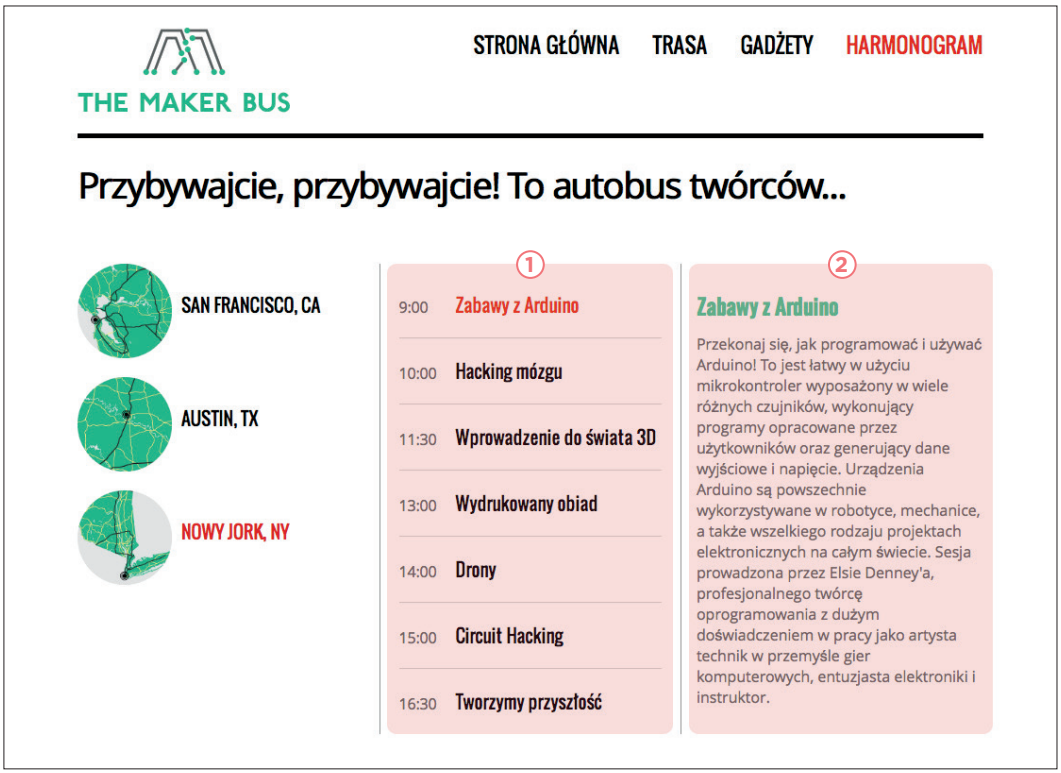
PRZYKŁAD
AJAX I JSON

Ten przykład wykorzystuje dane z trzech oddzielnych źródeł w celu zademonstrowania technik Ajax.

W lewej kolumnie znajdują się trzy miejsca, w których odbywają się pewne wydarzenia. Lokalizacje te są zdefiniowane w kodzie HTML dla strony zawierającej harmonogram wydarzeń. Każde wydarzenie jest łączem.

1. Kliknięcie zdarzenia powoduje wczytanie harmonogramu sesji dla danego zdarzenia. Informacje są przechowywane w pliku o nazwie *example.json*, który zostaje pobrany podczas wczytywania modelu DOM.

2. Kliknięcie sesji spowoduje wczytanie jej opisu. Opisy znajdują się w pliku *descriptions.html*, który jest wczytywany po kliknięciu tytułu sesji.



PRZYKŁAD
AJAX I JSON

HTML c08/example.html

```
<body>
  <header>
    <h1>THE MAKER BUS</h1>
    <nav>
      <a href="jq-load.html">STRONA GŁÓWNA</a>
      <a href="jq-load2.html">TRASA</a>
      <a href="jq-load3.html">GADŻETY</a>
      <a href="example.html" class="current">HARMONOGRAM</a>
    </nav>
  </header>
  <section id="content">
    <div id="container">
      <div class="third">
        <div id="event">
          <a id="ca" href="ca.html">
            <img src="img/map-ca.png" alt="SF, CA" />San Francisco, CA</a>
          <a id="tx" href="tx.html">
            <img src="img/map-tx.png" alt="Austin, TX" />Austin, TX</a>
          <a id="ny" href="ny.html">
            <img src="img/map-ny.png" alt="New York, NY" />Nowy Jork, NY</a>
        </div>
      </div>
      <div class="third">
        <div id="sessions">Wybierz wydarzenie w lewej kolumnie</div>
      </div>
      <div class="third">
        <div id="details">Informacje szczegółowe</div>
      </div>
    </div><!-- #container -->
  </section><!-- #content -->
  <script src="js/jquery-1.11.0.min.js"></script>
  <script src="js/example.js"></script>
</body>
```

W powyższym listingu znajduje się kod HTML strony. Na stronie mamy nagłówek oraz trzy kolumny. Wczytanie dwóch skryptów odbywa się przed znacznikiem zamykającym </body>.

Lewa kolumna: lista wydarzeń.
Środkowa kolumna: harmonogram sesji danego wydarzenia.
Prawa kolumna: opisy poszczególnych sesji.

PRZYKŁAD
AJAX I JSON

c08/data/example.json — JAVASCRIPT

```
{
    "CA": [
        {
            "time": "09.00",
            "title": "Wprowadzenie do świata 3D"
        },
        {
            "time": "10.00",
            "title": "Circuit Hacking"
        },
        {
            "time": "11.30",
            "title": "Zabawy z Arduino"
        }...
```

c08/descriptions.html — HTML

```
<div id="Wprowadzenie-do-modelowania-3D">
  <h3>Wprowadzenie do świata 3D</h3>
  <p>Przyjdź i przekonaj się, jak tworzyć modele 3D, które następnie...</p>
</div>
<div id="Circuit-Hacking">
  <h3>Circuit Hacking</h3>
  <p>W Electro-Tent będziesz mógł zobaczyć, jak...</p>
</div>
<div id="Zabawy-z-Arduino">
  <h3>Zabawy z Arduino</h3>
  <p>Przekonaj się, jak programować i używać Arduino! To jest łatwy...</p>
</div>
```

Kiedy skrypt zostaje uruchomiony, funkcja `loadTimetable()` wczytuje z pliku JSON o nazwie *example.json* harmonogramy wszystkich trzech wydarzeń. Dane są buforowane w zmiennej `times`.

Zdarzenia są identyfikowane przez dwuliterowy kod stanu w USA. Powyżej przedstawiono przykładowe dane sformatowane jako JSON oraz przykładowy kod HTML wygenerowany na podstawie wymienionych danych.

PRZYKŁAD
AJAX I JSON

JAVASCRIPT c08/js/example.js

```
1  $(function() {                              // Kiedy model DOM będzie gotowy.

2    var times;                                // Deklaracja zmiennej globalnej.
     $.ajax({                                  // Konfiguracja żądania.
       beforeSend: function(xhr){              // Przed wykonaniem żądania.
         if (xhr.overrideMimeType) {
         // Jeżeli przeglądarka obsługuje tę metodę,
3          xhr.overrideMimeType("application/json");
           // ustaw typ MIME, aby uniknąć błędów.
         }
       }
     });

     // Funkcja pobierająca dane z pliku JSON.
4    function loadTimetable() {                // Deklaracja funkcja
       $.getJSON('data/example.json')          // Próba zebrania danych JSON.
       .done( function(data){                  // Jeżeli zakończy się powodzeniem,
5        times = data;                         // to dane będą przechowywane w zmiennej.
       }).fail( function() {          // W przypadku problemu należy wyświetlić komunikat.
6        $('#event').html('Przepraszamy! Nie udało się wczytać harmonogramu.');
       });
     }

7    loadTimetable();                          // Wywołanie funkcji.
```

1. Skrypt wykonujący całą pracę ma nazwę *example.js* i jest uruchamiany po wczytaniu modelu DOM.

2. Zmienna `times` będzie użyta do przechowywania harmonogramu sesji wszystkich zdarzeń.

3. Zanim przeglądarka zażąda danych JSON, skrypt sprawdza, czy obsługuje ona metodę `overrideMimeType()`. Metoda ta jest stosowana w celu określenia, czy odpowiedź otrzymana z serwera powinna być traktowana jako dane JSON. Można ją wykorzystać, gdy serwer przypadkowo wskaże, że zwracane dane są w innym formacie.

4. Następnie mamy funkcję o nazwie `loadTimetable()` przeznaczoną do wczytania danych harmonogramów z pliku *example.json*.

5. Jeżeli wczytanie danych zakończy się powodzeniem, to dane harmonogramów będą przechowywane w zmiennej `times`.

6. Jeżeli wczytanie danych zakończy się niepowodzeniem, to użytkownikowi będzie wyświetlony komunikat o błędzie.

7. Funkcja `loadTimetable()` jest wywoływana w celu wczytania danych.

PRZYKŁAD
AJAX I JSON

c08/data/example.json — JAVASCRIPT

```
// Kliknięcie wydarzenia powoduje wczytanie
// harmonogramu.
$('#content').on('click', '#event a', function(e) {
// Użytkownik klika wybraną lokalizację.

  e.preventDefault();
  // Uniemożliwienie wczytania strony.
  var loc = this.id.toUpperCase();
  // Pobranie wartości atrybutu id.

  var newContent = '';
  // W celu utworzenia harmonogramu
  for (var i = 0; i < times[loc].length; i++) {
  // przeprowadzamy iteracje przez sesje.
    newContent += '<li><span class="time">' +
times[loc][i].time + '</span>';
    newContent += '<a href="descriptions.html#';
    newContent += times[loc][i].title.replace
(/ /g, '-') + '">';
    newContent += times[loc][i].title + '</a></li>';
  }

  $('#sessions').html('<ul>' + newContent + '</ul>');
  // Wyświetlenie godziny.

  $('#event a.current').removeClass('current');
  // Uaktualnienie wybranego łącza.
  $(this).addClass('current');

  $('#details').text('');
  // Usunięcie zawartości trzeciej kolumny.
});
```

1. Metoda pomocnicza jQuery czeka, aż użytkownik kliknie nazwę wydarzenia. W środkowej kolumnie wczytany będzie harmonogram sesji wybranego wydarzenia.

2. Metoda `preventDefault()` uniemożliwia przejście na nową stronę (ponieważ wyświetlone zostaną dane pobrane za pomocą technologii Ajax).

3. Utworzona zostaje zmienna o nazwie `loc`, przeznaczona do przechowywania miejsca, w którym odbywa się dane wydarzenie. Jej wartość jest pobierana z atrybutu `id` kliknięte łącza.

4. Kod HTML dla harmonogramu sesji będzie przechowywany w zmiennej o nazwie `newContent`. Początkowo wartością zmiennej jest pusty ciąg tekstowy.

5. Informacje o poszczególnych sesjach są przechowywane w elementach `<li>`. Na początku każdego elementu znajduje się godzina rozpoczęcia danej sesji.

6. Do harmonogramu zostaje dodane łącze używane w celu wyświetlenia opisu. Łącze prowadzi do pliku *descriptions.html*. Ponieważ łącze jest poprzedzone znakiem `#`, wskazuje ono odpowiedni fragment strony.

7. Tytuł sesji jest umieszczany po znaku `#`. Metoda `.replace()` zastępuje spacje w tytule łącznikami, aby dopasować wartość atrybutu `id` w pliku *descriptions.html* do poszczególnych sesji.

8. Wewnątrz łącza można dostrzec tytuł sesji.

9. Nowa zawartość jest umieszczana w środkowej kolumnie.

10. Atrybuty `class` w łączach wydarzeń są uaktualniane, aby wskazać bieżące wydarzenie.

11. Jeżeli trzecia kolumna zawierała jakiekolwiek dane, to zostają one usunięte.

PRZYKŁAD
AJAX I JSON

JAVASCRIPT c08/js/example.js

```
  // Kliknięcie sesji powoduje wczytanie jej opisu.
1 $('#content').on('click', '#sessions li a',
function(e) { // Kliknięcie sesji.
2   e.preventDefault();
    // Nie ma przejścia na nową stronę.
3   var fragment = this.href;
    // Tytuł znajduje się w atrybucie href.

4   fragment = fragment.replace('#', ' #');
    // Dodanie spacji po znaku #.
5   $('#details').load(fragment);
    // Wczytanie opisu.

    $('#sessions a.current').removeClass('current');
6   // Uaktualnienie elementu.
    $(this).addClass('current');
  });

  // Kliknięcie nawigacji.
  $('nav a').on('click', function(e) {
  // Kliknięcie w elemencie <nav>.
    e.preventDefault();
    // Nie ma przejścia na nową stronę.
    var url = this.href;
    // Pobranie adresu URL do wczytania.

7   $('nav a.current').removeClass('current');
    // Uaktualnienie klas w nawigacji.
    $(this).addClass('current');

    $('#container').remove();
    // Usunięcie starego elementu.
    $('#content').load(url + ' #container').hide().
fadeIn('slow'); // Dodanie nowego elementu.
  });
});
```

1. Inna metoda pomocnicza jQuery zostaje skonfigurowana w celu reakcji, gdy użytkownik kliknie sesję w środkowej kolumnie. Zadaniem metody jest wczytanie opisu sesji.

2. Metoda `preventDefault()` uniemożliwia przejście na nową stronę.

3. Utworzona zostaje zmienna o nazwie `fragment` przeznaczona do przechowywania łącza prowadzącego do sesji. Wartość tej zmiennej zostaje pobrana z atrybutu `href` kliknięte łącza.

4. Po znaku # umieszczona zostaje spacja, aby tym samym przygotować odpowiedni format dla metody jQuery `.load()`, odpowiedzialnej za pobranie fragmentu (nie całości) strony HTML, na przykład *description.html #Zabawy-z-Arduino*.

5. Selektor jQuery jest używany do znalezienia w trzeciej kolumnie elementu, którego atrybut `id` ma wartość `details`. Następnie metoda `.load()` wczytuje opis sesji w znalezionym elemencie.

6. Łącza zostają uaktualnione, aby podświetlić odpowiednią sesję w środkowej kolumnie.

7. Główna nawigacja na stronie zostaje ustawiona, jak pokazano wcześniej (patrz podrozdział „Wczytywanie zawartości”).

PODSUMOWANIE

- Ajax oznacza grupę technologii pozwalających na uaktualnienie tylko jednego fragmentu strony zamiast jej całej.
- Na stronie internetowej można wykorzystać dane w formatach HTML, XML i JSON. (Format JSON zdobywa coraz większą popularność).
- Aby wczytać dane JSON z innej domeny, można użyć JSONP, ale tylko wtedy, gdy kod pochodzi z zaufanego źródła.
- Biblioteka jQuery oferuje metody ułatwiające wykorzystanie technologii Ajax.
- Użycie metody `.load()` to najprostszy sposób umieszczenia zawartości HTML na stronie; pozwala na uaktualnienie tylko fragmentu strony.
- Metoda `.ajax()` oferuje znacznie większe możliwości, choć jest przy tym znacznie bardziej skomplikowana. (Dostępnych jest także kilka metod skrótów).
- Trzeba koniecznie rozważyć, jak będzie działała witryna, jeśli użytkownik ma wyłączoną obsługę JavaScript lub jeżeli strona nie ma dostępu do danych z serwera.

SKOROWIDZ

A

adres URL
  domena, 426
  port, 426
  protokół, 426
  skryptu, 361
  subdomena, 426
  względnego protokołu, 361
  względny, 395
Ajax, 374, 376, 377, 384, 386, 388, 390, 407
  obsługa błędów, 402
  żądanie, *Patrz:* żądanie Ajax
AngularJS, 434, 440, 441, 445
animacja, 340, 341, 362
API, 190, 416, 418
  Console, 476
  geolocation, 419, 420, 422, 423, 424
  history, 419, 430
  HTML, 419, 420
  localStorage, 419
  Map Google, 447, 448
  platform, 446
  sessionStorage, 419
  Web Storage, *Patrz:* Web Storage
aplikacja internetowa, 374
Application Programming Interface, *Patrz:* API
arkusz stylów, *Patrz:* CSS
atak XSS, 234
  ochrona, 235, 236, 237
atrybut, 14, 238, 239, 327
  alt, 556
  class, 8, 14, 72, 118, 193, 194, 195, 199, 205, 206, 211, 240, 291, 496, 497, 567
  id, 196, 198, 201
  placeholder, 600, 601
  required, 597, 606, 614
  src, 53
  tworzenie, 240, 326
  uaktualnienie, 195, 326
  usuwanie, 241, 326

B

biblioteka
  AngularJS, *Patrz:* AngularJS
  JavaScript, 366
  jQuery, *Patrz:* jQuery
  konflikt, 367
błąd, 456, 465, 466, 467, 468, 491, 616
  Ajax, 402
  komunikat, 616, 617
  obsługa, *Patrz:* wyjątek obsługa
  wyjątek, *Patrz:* wyjątek
  zgłaszanie, 488, 616, 617
    dla NaN, 489

C

CDN, 360, 435
Chrome, 242, 472, 474
ciąg tekstowy, 70, 71, 129, 134, 136, *Patrz też:* węzeł tekstowy
  konkatenacja, 84
  numer indeksu, 135
  operator, *Patrz:* operator ciągu tekstowego
Content Delivery Network, *Patrz:* CDN
cookies, 234, 426
CORS, 390
cross-site scripting, *Patrz:* atak XSS
CSS, 34, 46, 50
  reguła, 15, 46, 51
    selektor, *Patrz:* selektor
    właściwość, *Patrz:* właściwość
czas, 142, 143, 145

D

dane, 33
  dołączanie, 443
  filtrowanie, 534, 540, 541, 542, 549, 550, 554, *Patrz też:* filtr
    dynamiczne, 544, 548
    statyczne, 543
  JSON, *Patrz:* plik JSON
  niezaufane, 234
  pobieranie, 444
  przechowywanie w przeglądarce internetowej, 426
  sortowanie, 534, 560, 562
    dat, 565
    kolejność, 561, 564
    liczb, 564
    tabeli, 566, 570, 571
  typ, *Patrz:* typ
  wyszukiwanie, 534, 556
    live, 559
    tekstu, 558
data, 142, 143, 145
  sortowanie, 565
debugowanie, 26, 469, 471, 490
  stepping over, 483
  wejście do funkcji, 483
diagram, 24, 29, 152
DOM, 44, 127, 128, 129, 132, 190, 193, 228, 245, 303, 324, 416, 546
  drzewo modelu, 190, 192, 194, 224, 225, 229, 300
    zapytanie, *Patrz:* zapytanie
  obsługa zdarzeń, *Patrz:* procedura obsługi zdarzeń w modelu DOM
  używanie, 133
  w przeglądarce
    Chrome, 242
    Firefox, 242

E

ECMAScript, 538
element, 14, 218, 354, *Patrz też:* obiekt, węzeł
  dodawanie, 228, 229
  em, 218
  form, 400, 578
  kopiowanie, 352, 353
  odniesienie, 197, 546
  pobieranie zawartości, 320, 321
  położenie na stronie, 357, 358
  script, 53, 56, 57, 362, 363, 377, *Patrz też:* skrypt
  spinbox, 596
  table, 567
  uaktualnianie, 322
  usuwanie, 230, 231, 352, 353
  wstawianie, 324
  wybrany przez jQuery, *Patrz:* zbiór dopasowany
  wymiary, 354

F

Facebook, 446
filtrowanie, *Patrz:* dane filtrowanie
Firefox, 242, 473, 475
formularz sieciowy, 11, 309, 332, 348, 350, 574, 576, 614
  Ajax, 400
  hasło, *Patrz:* hasło
  HTML5, 596, 598
  jQuery, *Patrz:* jQuery UI formularz
  kontrolka, 578, 579
    select, 590
  lista rozwijana, 590
  pole
    tekstowe, 594
    wyboru, 582, 586
    zgody rodziców, 627, 628
  przycisk wysyłający formularz, 584
  strzałka, 596
  weryfikacja, 604, 606, 610, 611, 618, 620
    niestandardowa, 624

zgody rodziców, 627, 628
wysyłanie, 580, 609
funkcja, 92, 94, 137
anonimowa, 94, 102, 103, 254, 262
argument, 99, 103
deklaracja, 96, 98, 102
globalna, 120
identyfikator, *Patrz:* funkcja nazwa
IIFE, 103, 610
jQuery, 303
konstruktora, 114, 115, 116, 117
lokalna, 120
nazwa, 94, 96
nazwana, 102, 107, 254
parametr, 94, 98, 99
pomocnicza, 576, 577
porównująca, 561, 566, 568
trygonometryczna, 140
wartość zwrotna, 94
wykrywacz, 307
wynik, 100
wyrażenia, 102, 103
jako metoda, 121
wywołania zwrotnego, 340
wywoływanie, 94, 95, 97, 99
yepnope, 602, 603
zagnieżdżanie, 121
zakres leksykalny, 463

G

geolokalizacja, *Patrz:* API geolocation
Google, 446
Google mapa, 447, 448
kontrolki, 450
styl, 452
znacznik, 453

H

hasło, 583, 625, 632, 633
HTML, 46, 50, 380
API, *Patrz:* API HTML
element, *Patrz:* element
HTML5 formularz sieciowy, 596, 598
HTML5 Storage, *Patrz:* Web Storage

I

interfejs
programowania aplikacji, *Patrz:* API
użytkownika, 416
Internet Explorer, 16, 53, 215, 222, 223, 256, 259, 261, 264, 265, 268, 270, 306, 307, 379, 576
interpreter, 47, 459, 464
iteracja niejawna, 316

J

JavaScript, 50, 51, 60
HISTORIA, 538
interpreter, *Patrz:* interpreter
kod
blok, 62, 94
dołączanie, 53
tworzenie, 52, 62
źródłowy, 54, 87
komentarz, *Patrz:* komentarz
konsola, 470, 476, 477, 478
Chrome, 472, 474
Firefox, 473, 475
obsługa błędów, 456
polecenie, *Patrz:* polecenie
JavaScript Object Notation, *Patrz:* JSON
język
CSS, *Patrz:* CSS
HTML, *Patrz:* HTML
interpretowany, 47
JavaScript, *Patrz:* JavaScript
składnia, 26, 60
jQuery, 16, 195, 215, 234, 256, 300, 302, 303, 304, 306, 307, 369, 374, 390, 394, 416, 537
dokumentacja, 364
dołączanie, 360
z CDN, 361
efekty, 338, 339, 340
funkcja, *Patrz:* funkcja jQuery

jQuery
metoda, *Patrz:* metoda jQuery
obiekt, *Patrz:* obiekt jQuery
pętla, *Patrz:* pętla jQuery
rozszerzenia, 365
skrót metody Ajax, 398
UI, 435
accordion, 436
formularz, 435
karta, 437
wersja, 307
właściwość, *Patrz:* właściwość jQuery
wtyczka, 365, 434, 494
accordion, 529, 530
tworzenie, 528
UI, *Patrz:* jQuery UI
zmienna, *Patrz:* zmienna jQuery
JSON, 374, 378, 380, 382, 388, 407
JSONP, 390, 391, 392

K

klucz, 107, 108, 122
JSON, 382
klucz-wartość, 107, 123, 124, 426
kolekcja
elementów, 578
formularzy, 578
komentarz, 63
komponent nasłuchujący, 103
kontekst wykonywania, 459, 462
obiekt zmiennych, 462

L

liczba
całkowita, 138
dziesiętna, 139
losowa, 141
pi, 140
rzeczywista, 138
sortowanie, 564
wykładnicza, 138
zaokrąglenie, 138, 139, 140
zmiennoprzecinkowa, 138
lightbox, *Patrz:* okno modalne
lista, 76
numerowana, 78
rozwijana, 590, 591
livesearch, 555, 559
logika
prywatna, 507
publiczna, 507

Ł

łącze, 497

M

metoda, 34, 38, 56, 92, 106, 107
$.ajax, 394, 402, 404, 405
$.get, 394, 402
$.getJSON, 394, 402
$.getScript, 394
$.isNumeric, 349
$.post, 394, 400, 402
.abort, 395
.add, 344, 537
.addClass, 326
.after, 324, 325, 351
.always, 395, 402, 403
.animate, 338, 340, 341, 499
.append, 321, 324
.appendTo, 324
.attr, 326
.before, 324, 325
.children, 342
.clone, 352, 353
.closest, 342
.complete, 402
.css, 328, 329
.data, 608, 616
.delay, 338, 339
.detach, 352
.done, 395, 402, 403
.each, 316, 330, 331, 339, 345, 537
.empty, 352

.eq, 346, 347
.error, 402
.fadeIn, 338
.fadeOut, 338
.fadeTo, 338
.fadeToggle, 338
.fail, 395, 402, 403
.filter, 344, 345, 349, 537
.find, 342, 344
.has, 344, 345
.height, 354, 355, 356
.hide, 338, 499
.html, 320, 321, 322
.innerHeight, 354
.innerWidth, 354
.is, 344, 345, 349
.load, 319, 394, 395, 396
.next, 342
.nextAll, 342
.noConflict, 367
.not, 344, 537
.offset, 357
.on, 319, 332, 336, 349
.outerHeight, 354
.outerWidth, 354
.parent, 342
.parents, 342
.position, 357
.post, 400
.prepend, 324, 325
.prependTo, 324
.prev, 342
.prevAll, 342
.preventDefault, 334, 351
.ready, 318, 319, 367
.remove, 322, 352
.removeAttr, 326
.removeClass, 326
.replaceWith, 322
.scrollLeft, 356
.scrollTop, 356
.serialize, 349, 400, 401
.show, 338, 499
.siblings, 342
.slideDown, 338, 339
.slideToggle, 338
.slideUp, 338
.stop, 338
.stopPropagation, 334
.success, 402
.text, 320, 321, 322
.toArray, 537
.toggle, 338, 499
.unwrap, 352
.val, 349, 351
.width, 354, 355, 356
add, 590
addEventListener, 261, 264, 265, 274
appendChild, 195, 228
attachEvent, 264, 274
blur, 579
className, 195
clear, 427
click, 579
concat, 536
console.assert, 481
console.error, 478
console.group, 479
console.groupEnd, 479
console.info, 478
console.log, 476
console.table, 480
console.warn, 478
createElement, 195, 228, 229
createTextNode, 195, 228
e.PreventDefault, 401
every, 536
filter, 536, 542
floor, 140, 141
focus, 579
forEach, 536, 542
getAttribute, 195, 238, 239
getCurrentPosition, 423, 424
getElementById, 194, 196, 198, 199, 200, 202, 241

metoda
getElementsByClassName, 194, 199, 202, 203
getElementsByTagName, 194, 199, 202, 203, 207
getFullYear, 144
getItem, 427, 428
globalna, 349
hasAttribute, 195, 238, 239
history.back, 432
history.forward, 432
history.go, 432
history.pushState, 432
history.replaceState, 432
html, 237
indexOf, 556
item, 204, 205
jQuery, 302, 313
JSON.parse, 383, 388
JSON.stringify, 383
łączenie, 317
map, 536
obiektu, 121
pop, 536
preventDefault, 268, 273, 274, 500
push, 536
pushState, 430, 431
querySelector, 194, 199, 200, 202, 208, 306
querySelectorAll, 194, 199, 202, 203, 208, 306
random, 140, 141
remove, 590
removeAttribute, 195, 238
removeChild, 195, 230
removeEventListener, 261
removeItem, 427
replaceState, 430, 431
reverse, 536
select, 579
send, 379, 385
setAttribute, 195, 238
setItem, 427
shift, 536
skrótów, 398
some, 536
sort, 536, 560, 561
stopPropagation, 268, 273
text, 237
toDateString, 143
toTimeString, 143
unshift, 536
window.alert, 130
window.open, 130
window.print, 130
write, 56, 232, 361
zdarzeń, 332
minimalizacja, 304
model
obiektowy, *Patrz:* DOM
przetwarzania
asynchronicznego, 377
synchronicznego, 377
Modernizr, 420, 421, 599, 602
MVC, 440, 442

N

nazwa-wartość, 34, 94
NodeList, 198, 202, 210, 211
element, 204, 206
statyczna, 202
typu live, 202
notacja
literału, *Patrz:* obiekt notacja literału
nawiasu kwadratowego, *Patrz:* obiekt notacja nawiasu kwadratowego
wykładnicza, 138
z użyciem konstruktora, *Patrz:* obiekt notacja z użyciem konstruktora
z użyciem kropki, *Patrz:* obiekt notacja z użyciem kropki

O

obiekt, 56, 92, 107, 123, *Patrz też:* element, węzeł
  Array, *Patrz:* tablica
  cache, 515
  console, 476
  Date, 142, 143, 144
  document, 42, 44, 46, 56, 132
    właściwość, 44, 132
    zdarzenie, 44
  domyślny, 131
  dostęp za pomocą nazwy, 539
  egzemplarz, 115, 117, 142
  error, 268, 465
  event, 268, 271, 273, 274, 276, 334, 335
  globalny, 127, 129, 134, 138, 140, 143
  history, 430, 431, 432
  jako właściwości, 539
  jQuery, 303, 314, 315
  jqXHR, 395
  kolejność, 539
  localStorage, 426, 428, 429
  location, 431
  Math, 140
  modal, 510
  notacja
    literału, 108, 110, 111, 119
    nawiasu kwadratowego, 109
    z użyciem konstruktora, 108, 114, 116, 119, 123
    z użyciem kropki, 109, 118
  Number, 138
  opakowania, *Patrz:* obiekt String
  pamięci masowej, 426
  potomny, 131
  serializacja, *Patrz:* serializacja
  sessionStorage, 426, 428, 429
  String, 134, 135, 556
  tworzenie, 108, 110, 111, 112, 114, 116, 117, 119, 123
  uaktualnienie, 113
  valid, 610
  variables, 463
  w stylu tablicy, 346
  w tablicy, 125, 127, 539
  wbudowany, 92, 126, 128
  window, 42, 130, 254, 259
  właściwość, *Patrz:* właściwość
  XDomainRequest, 390
  XMLHttpRequest, 378, 379, 385
obiektowy model dokumentu, *Patrz:* DOM
obraz, 550, 554, 556
obserwator zdarzeń, *Patrz:* procedura obsługi zdarzeń obserwator
obsługa błędów, *Patrz:* wyjątek obsługa
okno
  modalne, 495, 506
    skrypt, 509
    tworzenie, 508
  wymiary, 356
operator, 81
  arytmetyczny, 81, 82, 83
  ciągu tekstowego, 81, 84, 85
  elementu składowego, 109, 113
  grupowania, 103
  jednoargumentowy, 174
  kolejność, 81
  logiczny, 81, 162, 163, 175
    AND, 163, 164
    NOT, 163, 165
    OR, 163, 165
  porównania, 81, 154, 156, 157, 159, 160
    struktura, 158
  przypisania, 81

P

panel zawartości, 494, 495
  accordion, 495, 497, 498, 529, 530
    tworzenie, 500
  kart, 495, 497, 502
    tworzenie, 504
  okno modalne, *Patrz:* okno modalne
  przeglądarka zdjęć, *Patrz:* przeglądarka zdjęć
  slajdy, *Patrz:* slajdy
pasek
  adresu, 431
  przewijania, 356

pętla, 176, 180, 185, 210
do-while, 176, 183
działająca w nieskończoność, 180
for, 176, 177, 181
jQuery, 316
licznik, 176, 177
while, 176, 182
plik
console-log.html, 476
cookies, *Patrz:* cookies
CSS, 50, 360
error.html, 471
HTML, 50, 360, 374, 378, 380
JS, 50, 360
JSON, *Patrz:* JSON
utilities.js, 577, 580
XML, 374, 378, 380, 381, 386
pokaz slajdów, *Patrz:* slajdy
pole wyboru, *Patrz:* formularz sieciowy pole wyboru
polecenie, 62
catch, 468, 486, 487
finally, 468, 486, 487
pętli, *Patrz:* pętla
throw, 468, 488
try, 468, 486, 487
warunkowe, 155
if, 155, 166, 167
if-else, 168, 169, 185
switch, 155, 170, 171
procedura
kolejność przetwarzania, 458
obsługi zdarzeń, 103, 254, 256, 307, 332
obserwator, 256, 261, 263, 269, 271
parametr, 262, 263, 269, 336
w HTML, 256, 257
w modelu DOM, 256, 259
progressive enhancement, *Patrz:* strona stopniowe ulepszanie
proxy, 390
przeglądarka internetowa, 42, 44, 46, 307
Chrome, *Patrz:* Chrome
Firefox, *Patrz:* Firefox
Internet Explorer, *Patrz:* Internet Explorer
przechowywanie danych, *Patrz:* dane przechowywanie w przeglądarce internetowej
silnik, 46
skryptowy, *Patrz:* interpreter
wersja, 16
przeglądarka zdjęć, 495, 512, 514, 515
przetwarzanie
asynchroniczne, 377
nieblokujące, *Patrz:* przetwarzanie asynchroniczne
przycisk, 497
filtrowania, 555
opcji, 588
wysyłający formularz, 584
pudełko, 354, 499, *Patrz też:* element
punkt kontrolny, 469, 482, 483
warunkowy, 484

S

selektor, 15, 200, 300, 302, 306, 345, 346, 348
:button, 348
:checkbox, 348
:checked, 348
:contains, 344, 345
:disabled, 348
:enabled, 348
:file, 348
:focus, 348
:gt, 346, 347
:has, 344
:image, 348
:input, 348
:lt, 346, 347
:not, 344, 345
:password, 348
:radio, 348
:reset, 348
:selected, 348
:submit, 348
:text, 348, 351

class, 496
filtr, 308, 344
atrybutu, 309
potomny, 309
widoczności, 309
zawartości, 309, 310
formularz sieciowy, 309
hierarchii, 308
podstawowy, 308
serializacja, 388
serwer WWW, 378, 390
skrypt, 20
adres URL, 361
angular.js, 440
jQuery, 304
polyfill, 599, 601, 602, 603
tworzenie, 22, 27, 28
umieszczanie na stronie, 55
slajdy, 10, 495, 497, 520, 522, 523
słowo
kluczowe, 75
break, 170, 180
case, 170
continue, 180
debugger, 485
delete, 113, 118
function, 96
new, 115
null, 379
this, 114, 119, 121, 259, 276, 330
var, 66
zarezerwowane, 75
sortowanie, *Patrz:* dane sortowanie
stos, 460
strona, 56
model, 192
odświeżenie, 11
przeglądanie, 430
stopniowe ulepszanie, 51
struktura, 46
wymiary, 356

T

tablica, 76, 78, 101, 107, 122, 124, 137, 180, 205, 314, 536, 538, 560
długość, 78
element, 536
dodawanie, 536
filtrowanie, 536, 549
iteracja, 536
łączenie, 536
modyfikacja, 536
usuwanie, 536
zmiana kolejności, 536
indeks, 78, 122
konstruktor, 77
literał, 77
nazwa, 77
obiektów, 125, 127
sortowanie, 565, 570, 571
tworzenie, 77
wyświetlanie, 541
tag, 550, 551
ThemeRoller, 435
token, 234
Twitter, 446
typ
array, 382
Boolean, 68, 72, 129, 172, 382
liczbowy, *Patrz:* typ liczbowy
narzucanie, 172
Null, 137, 172, 382
Number, 68, 129, 138, 172, 382
object, 382
określanie typu, 172
String, 68, 129, 172, 382
tekstowy, *Patrz:* typ String
Undefined, 137, 172
weryfikacja, 620

U

usługa $http, 444

W

warstwa
  HTML, *Patrz:* warstwa zawartości
  prezentacyjna, 50, 51
  zachowania, 50
  zawartości, 50, 51
wartość
  falsy, 173, 174
  NaN, 84, 174
    błąd, 489
  truthy, 173, 174
  undefined, 80, 174
Web Storage, 426, 427
weryfikacja, 235, 236
węzeł, 46, 192, 312, *Patrz też:* element, obiekt
  atrybutów, 192, 193, 238
  document, 192, 193
  elementu, 192, 193, 194, 238
  kolekcja, *Patrz:* NodeList
  nadrzędny, 214, 219
  odniesienie, 314, 315, 546
  położenie, 197
  potomny, 214, 217, 226
  równorzędny, 216
  tekstowy, 192, 193, 195, 215, 218, 220, 221
widżet, 435
właściwość, 15, 34, 35, 44, 103, 106, 107, 109, 328
  cancelable, 268
  cancelBubble, 268, 273
  className, 216, 238, 240
  CSS, 328, 329, 340
  data, 334
  document, 130
  dodawanie, 113, 118
  firstChild, 194, 214, 215, 217
  history, 130
  id, 238
  innerHeight, 130, 131
  innerHTML, 133, 219, 224, 226, 233, 234, 235, 237
  innerText, 219, 222, 223, 237
  innerWidth, 130, 131
  jQuery, 302
  JQXHR, 395
  lastChild, 194, 214, 215, 217
  location, 130
  nextSibling, 194, 214, 215, 216
  nodeValue, 195, 220
  pageX, 334
  pageXOffset, 130
  pageY, 334
  pageYOffset, 130
  parentNode, 194, 214
  previousSibling, 194, 214, 215, 216
  returnValue, 268, 273
  screen, 130
  screenX, 130
  screenY, 130
  srcElement, 268
  target, 268, 276, 334
  textContent, 195, 219, 222, 223, 237
  timeStamp, 334, 335
  type, 268, 334, 335
  usuwanie, 113, 118
  which, 334
wyciek pamięci, 420
wyjątek, 464, 472
  obsługa, 464, 486, 608
wyrażenie, 80, 102, 160
  porównywanie, 161
  regularne, 620, 622, 623
wyszukiwanie, *Patrz:* dane wyszukiwanie
wzorzec projektowy, 507

X

XML, *Patrz:* plik XML

Y

YepNope, 602, 603

Z

zapytanie, 196, 202, 203
zbiór dopasowany, 312, 330, 344
zdarzenie, 9, 34, 36, 44, 250, 277, 303, 332
  aktywności, 253
  blur, 253, 255, 332, 349, 579
  change, 253, 332, 349, 579
  click, 252, 332, 579
  copy, 253
  cut, 253
  dblclick, 252, 332
  delegacja, 272, 337
  dołączenie, 254, 256
  DOMContentLoaded, 319
  DOMNodeInserted, 253
  DOMNodeInsertedIntoDocument, 253
  DOMNodeRemoved, 253
  DOMNodeRemovedFromDocument, 253
  DOMSubtreeModified, 253
  error, 252, 332
  focus, 253, 332, 349, 579
  focusin, 253
  focusout, 253
  hover, 332
  input, 253, 332, 579, 559
  jQuery, 332
  keydown, 252, 332, 579
  keypress, 252, 332, 579
  keyup, 252, 332, 579
  klawiatury, 252, 332
  load, 252, 319, 332, 362
  modelu DOM, 254
  mousedown, 252, 332
  mousemove, 252, 332
  mouseout, 252, 332
  mouseover, 252, 254, 332
  mouseup, 252, 332
  myszy, 252, 332
  obserwator, *Patrz:* procedura obsługi zdarzeń obserwator
  onload, 379, 385
  paste, 253
  procedura obsługi, *Patrz:* procedura obsługi zdarzeń
  propagacja, 266, 267
  przechwytywanie, 266, 267
  przepływ, 266, 267
  resize, 252, 332
  scroll, 252, 332, 359
  select, 253, 332, 349
  submit, 253, 254, 255, 332, 349, 581
  UI, 252, 254, 332
  unload, 252, 332
zmienna
  $listItems, 315
  64, 69, 80, 122
  deklarowanie, 66
  globalna, 104, 105, 120
  jako część obiektu, 106
  jQuery, 315
  lokalna, 104, 105
  na poziomie funkcji, 104
  nazwa, 66, 67, 75, 106
    konflikt, 103, 105
  skrót, 73
  typ, *Patrz:* typ
  undefined, 80
  zakres, 104, 459, *Patrz też:* zmienna zasięg
  zasięg, 67, *Patrz też:* zmienna zakres
znacznik
  kontrola, 237
  otwierający, 14
  script, 236
  zamykający, 14, 304, 318
znak
  −−, 81
  !, 163
  !=, 156, 174
  !==, 156, 174
  &&, 162, 163
  *, 308
  ||, 163
  +, 84

znak
++, 82
==, 154, 174
===, 156, 174
apostrof, 70, 71, 382
cudzysłów, 70, 71, 227, 382
cytowania, 69, 70, 71, 84, 227
dolara, 75, 302, 330, 367
kropki, 75
myślnika, 75
odstępu, 215
podkreślenia, 75
specjalny, 71
ukośnika, 71, 227

Ż

żądanie
Ajax, 306, 379, 384, 394, 404
obsługa, 379
danych, 399


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

KOMPENDIUM WIEDZY NA TEMAT JAVASCRIPTU I JQUERY!

JavaScript to język, który w dużej mierze ukształtował współczesne strony WWW. Dzięki niemu możemy swobodnie korzystać z interaktywnych, wygodnych w użyciu oraz niezawodnych aplikacji internetowych. Pojawienie się JavaScriptu pozwoliło zastąpić tradycyjne aplikacje desktopowe nowymi, pracującymi w chmurze. Wokół języka powstało już wiele narzędzi i bibliotek. Jedną z najpopularniejszych jest jQuery.

Jeżeli chcesz poznać potencjał tego duetu i zacząć tworzyć atrakcyjne aplikacje internetowe, nie możesz obejść się bez tej książki. Pomoże Ci ona szybko stworzyć pierwszy skrypt. W trakcie lektury poznasz niuanse składni JavaScriptu, sposoby obsługi zdarzeń oraz obiektowy model strony. Dzięki dalszym rozdziałom zdobędziesz wiedzę na temat jQuery oraz możliwości tej biblioteki. Z pomocą duetu JavaScript i jQuery błyskawicznie rozwiążesz każdy problem — asynchroniczne pobieranie danych z serwera, atrakcyjny interfejs użytkownika, zaawansowana obsługa formularzy to tylko niektóre z poruszanych tu tematów. Książka ta jest doskonałym źródłem informacji dla czytelników chcących opanować JavaScript oraz związane z nim narzędzia!

SPRAWDŹ:

- jak łatwo wykorzystać potencjał JavaScriptu
- jak manipulować drzewem DOM
- jak obsługiwać zdarzenia
- jak pobierać dane z serwera
- jak tworzyć atrakcyjne aplikacje internetowe

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-8322-755-9

9 788383 227559

Cena: 129,00 zł