AJAX

Marcin Lis

TABLICE INFORMATYCZNE

Helion

OBSŁUGA TRANSMISJI

Umieszczanie skryptów w kodzie

Domyślny typ skryptów

Dokument HTML powinien zawierać w nagłówku określenie domyślnego języka skryptowego, który będzie w nim używany. Nie jest to konstrukcja obligatoryjna, ale zalecana przez konsorcjum standaryzacyjne W3C. Należy wykorzystać znacznik `<meta>` o schematycznej postaci:

```
<meta http-equiv="Content-Script-Type" content="typ">
```

gdzie *typ* określa rodzaj języka.

W przypadku JavaScript konstrukcja ta powinna mieć postać:

```
<meta http-equiv="Content-Script-Type" content="text/
↪javascript">
```

Jeśli powyższy znacznik nie zostanie umieszczony w kodzie, przeglądarka podejmie próbę ustalenia domyślnego języka skryptowego na podstawie otrzymanych z serwera nagłówków HTTP.

Osadzanie skryptów w kodzie HTML

Umieszczenie skryptu w kodzie HTML wymaga użycia znacznika `<script>`, schematycznie:

```
<script type="typ skryptu">
//kod skryptu
</script>
```

Formalnie parametr `type` jest obligatoryjny (począwszy od specyfikacji HTML 4.0) i powinien wskazywać na język skryptu; w przypadku skryptów JavaScript powinien to być ciąg znaków `text/javascript`, np.:

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

Większość dostępnych na rynku przeglądarek zaakceptuje jednak również znacznik `<script>` niezawierający tego argumentu.

Skrypty w zewnętrznych plikach

Jeżeli kod skryptu ma się znajdować w zewnętrznym pliku, należy użyć parametru `src` znacznika `<script>`, schematycznie:

```
<script type="typ skryptu" src="adres skryptu"><
↪/script>
```

np.:

```
<script type="text/javascript" src="http://nazwa.domeny/
↪skrypty/skrypt.js"></script>
```

Jeżeli plik ze skryptem znajduje się w tej samej lokalizacji co dokument HTML, wystarczy podanie samej nazwy, np.:

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

Należy pamiętać, że nie można pominąć znacznika zamykającego `</script>`.

Popularne przeglądarki obsługujące AJAX

Nazwa przeglądarki	Minimalna wersja
Firefox	1.0
Internet Explorer	5.0
Netscape	7.0
Opera	7.6
Safari	1.2

Obiekt XMLHttpRequest

Metody obiektu XMLHttpRequest

W tabeli zostały użyte następujące oznaczenia: FF — Firefox, IE — Internet Explorer, OP — Opera.

Nazwa metody	Znaczenie	Dostępność
`abort`	Przerywa żądanie HTTP	FF, IE, OP
`getAllResponse-Headers`	Pobiera nagłówki HTTP w postaci ciągu znaków	FF, IE, OP
`getResponseHeader`	Pobiera wybrany nagłówek HTTP	FF, IE, OP
`open`	Inicjuje żądanie HTTP	FF, IE, OP
`send`	Wysyła żądanie HTTP do serwera	FF, IE, OP
`setRequestHeader`	Ustawia wybrany nagłówek HTTP	FF, IE, OP
`overrideMimeType`	Nadpisuje nagłówek MIME zwrócony przez serwer	FF, IE, OP

Właściwości obiektu XMLHttpRequest

Nazwa właściwości	Znaczenie	Dostępność
`channel`	Określa kanał użyty do obsługi żądania	FF
`onreadystatechange`	Procedura obsługi zdarzenia wywoływana podczas zmiany właściwości `readyState`	FF, IE
`readyState`	Kod określający status żądania	FF, IE, OP
`responseBody`	Odpowiedź serwera w postaci tablicy bajtów	IE
`responseStream`	Odpowiedź serwera w postaci nieprzetworzonego strumienia bajtów	IE
`responseText`	Odpowiedź serwera w postaci tekstowej	FF, IE, OP
`responseXML`	Odpowiedź serwera w postaci XML	FF, IE, OP
`status`	Kod odpowiedzi serwera	FF, IE, OP
`statusText`	Kod odpowiedzi serwera w postaci tekstowej	FF, IE, OP

Tworzenie obiektów

Tworzenie bezpośrednie

Przeglądarki Firefox, Netscape, Opera, Safari czy Konqueror, a także Internet Explorer, począwszy od wersji 7., obsługują obiekty typu `XMLHttpRequest` bezpośrednio. W ich przypadku można użyć konstrukcji w postaci:

```
var zmienna = new XMLHttpRequest();
```

W przypadku Internet Explorera w wersjach poniżej 7. obsługa `XMLHttpRequest` jest możliwa jedynie przez kontrolki ActiveX:

```
var zmienna = new ActiveXObject("Microsoft.XMLHTTP");
```

Aby rozpoznać daną sytuację, należy zbadać, czy istnieją właściwości: `window.XMLHttpRequest` — co jest charakterystyczne dla wszystkich przeglądarek oprócz Internet Explorera w wersjach poniżej 7. `window.ActiveXObject` — co jest charaktery styczne dla Internet Explorera w wersjach poniżej 7.

Ogólna konstrukcja kodu ma postać:

```
if (window.XMLHttpRequest){
  //instrukcje tworzące obiekt bezpośrednio
}
else if (window.ActiveXObject){
  //instrukcje tworzące obiekt za pomocą ActiveX
}
```

Użycie wyjątków

Instrukcje, które mogą spowodować błąd, powinny być ujęte w blok `try...catch` o postaci:

```
try{
  //instrukcje mogące spowodować wyjątek
}
catch(identyfikatorWyjątku){
  //obsługa wyjątku
}
```

Bloki `try...catch` mogą być zagnieżdżane, zatem kolejny blok `try...catch` można umieścić w nawiasie klamrowym występującym zarówno po `try`, jak i po `catch`, np.:

```
try{
  //instrukcje mogące spowodować wyjątek
}
catch(identyfikatorWyjątku1){
  try{
    //instrukcje mogące spowodować wyjątek
  }
  catch(identyfikatorWyjątku2){
    //obsługa wyjątku 2
  }
}
```

lub też:

```
try{
  //instrukcje mogące spowodować wyjątek 1
  try{
    //instrukcje mogące spowodować wyjątek 2
  }
  catch(identyfikatorWyjątku2){
    //obsługa wyjątku 2
  }
}
catch(identyfikatorWyjątku1){
}
```

Aby użyć tej techniki do utworzenia obiektu typu `XMLHttpRequest` niezależnie od tego, z jakim typem przeglądarki mamy do czynienia, należy użyć konstrukcji:

```
var XMLHttpRequestObject = null;

try{
  XMLHttpRequestObject = new XMLHttpRequest();
}
catch(e){
  try{
    XMLHttpRequestObject = new ActiveXObject("Microsoft.
↪XMLHTTP");
  }
  catch(e){
  }
}
```

Po wykonaniu tych instrukcji zmienna `XMLHttpRequestObject` będzie zawierała albo obiekt typu `XMLHttpRequest`, albo wartość `null`, co można zbadać za pomocą instrukcji warunkowej `if`, np.:

```
if(XMLHttpRequestObject){
  alert("Obiekt XMLHttpRequest został utworzony.");
}
else{
  alert("Obiekt XMLHttpRequest nie został utworzony.");
}
```

Można też jawnie badać stan zmiennej `XMLHttpRequestObject`:

```
if(XMLHttpRequestObject == null){
```

Użycie kontrolek ActiveX

W przypadku korzystania z ActiveX zamiast kontrolki `Microsoft.XMLHTTP` można również użyć jej nowszych wersji: `MSXML2.XMLHTTP`, `MSXML2.XMLHTTP.3.0`, `MSXML2.XMLHTTP.4.0`, `MSXML2.XMLHTTP.5.0`, `MSXML2.XMLHTTP.6.0`. Aby użyć możliwie najnowszej z nich, można skorzystać z serii instrukcji warunkowych, serii bloków `try...catch` lub tablicy i pętli. Najkorzystniejszą i najbardziej skondensowaną wersją kodu jest ostatnia z wymienionych; struktura miałaby postać:

```
var XMLHttpRequestObject = null;
var wersjeMSXML = new Array(
  "MSXML2.XMLHTTP.6.0",
  "MSXML2.XMLHTTP.5.0",
  "MSXML2.XMLHTTP.4.0",
  "MSXML2.XMLHTTP.3.0",
  "MSXML2.XMLHTTP",
  "Microsoft.XMLHTTP"
  );

for (var indeks in wersjeMSXML){
  try{
    XMLHttpRequestObject = new ActiveXObject(wersjeMS
↪XML[indeks]);
    break;
  }
  catch(e){
  }
}
```

W przypadku korzystania z osobnych bloków `try...catch` najwygodniej jest umieścić je w oddzielnej funkcji zwracającej obiekt typu `XMLHttpRequest`, np.:

```
if(window.XMLHttpRequest){
  return new XMLHttpRequest();
}
else if(window.ActiveXObject){
  try{
    return new ActiveXObject("MSXML2.XMLHTTP.6.0");
  }
  catch(err){
    //
  }
  try{
    return new ActiveXObject("MSXML2.XMLHTTP.5.0");
  }
  catch(err){
    //
  }
  try{
    return new ActiveXObject("MSXML2.XMLHTTP.4.0");
  }
  catch(err){
    //
  }
  try{
    return new ActiveXObject("MSXML2.XMLHTTP.3.0");
  }
  catch(err){
    //
  }
  try{
    return new ActiveXObject("MSXML2.XMLHTTP");
  }
  catch(err){
    //
  }
  try{
    new ActiveXObject("Microsoft.XMLHTTP");
  }
  catch(err){
    //
  }
  return null;
}
else{
  return null;
}
```

Obsługa transmisji

Wysyłanie żądań do serwera

Aby zainicjować żądanie pobrania danych z serwera, należy wywołać metodę `open` obiektu `XMLHttpRequest`. Pozwala to na określenie adresu URL, do którego ma nastąpić odwołanie, a także parametrów transmisji. Ogólna deklaracja metody `open` ma postać:

```
open(metoda_transmisji, url, async, użytkownik,
↪ hasło)
```

Znaczenie poszczególnych argumentów jest następujące:

- *`metoda_transmisji`* — metoda transmisji danych,
- *`url`* — adres pliku lub skryptu umieszczonego na serwerze,
- *`async`* — określenie, czy transmisja ma być asynchroniczna,
- *`użytkownik`* — nazwa użytkownika (o ile dostęp do serwera wymaga autoryzacji),
- *`hasło`* — hasło użytkownika (o ile dostęp do serwera wymaga autoryzacji).

W przypadku metody transmisji należy skorzystać ze standardowych typów, jakie wykorzystuje się np. przy przesyłaniu danych z formularzy HTML. Mogą to więc być: `GET`, `POST`, `PUT`. Najczęściej używa się metod `GET` i `POST`.

Argument *`url`* może być podany w postaci bezwzględnej lub względnej. W pierwszym przypadku należy podać pełną ścieżkę dostępu do pliku, którego dotyczy żądanie, np.:

```
http://mojadomena.com/dane/dane.txt
```

W drugim przypadku jedynie ścieżkę dostępu do pliku, bez określania serwera, np.:

```
/dane/dane.txt
```

co oznacza odwołanie się do pliku znajdującego się na tym samym serwerze co skrypt. Jeśli plik, do którego ma nastąpić odwołanie, będzie w tym samym katalogu co skrypt, można całkowicie zrezygnować z podawania ścieżki dostępu i w takim przypadku wartością parametru *`url`* może być sama nazwa pliku, np.:

```
dane.txt
```

Argument *`async`* może być ustawiony na `true` lub `false`. Wartością domyślną jest `true`, co oznacza asynchroniczną transmisję danych. Dzięki temu po wysłaniu żądania do serwera skrypt nie będzie oczekiwał na transmisję danych, ale kontynuował działanie. Argumenty *`użytkownik`* i *`hasło`* są opcjonalne i należy ich użyć tylko w sytuacji, kiedy dostęp do serwera wymaga autoryzacji. Przykładowe wywołania metody `open`:

```
open("GET", "http://mojadomena.com/dane.txt");
open("POST", "http://mojadomena.com/dane.php", false);
open("POST", "http://mojadomena.com/dane.php", true,
↪ "user", "pass");
```

Do wysyłania danych do serwera używana jest metoda `send`. Jej wywołanie ma postać:

```
send(dane)
```

Jeżeli metodą transmisji jest `GET`, dane znajdują się w adresie URL użytym w wywołaniu `open`. Argument *`dane`* jest więc pomijany, a zamiast niego zostaje zastosowana wartość `null`:

```
send(null)
```

Transmisja asynchroniczna

W przypadku transmisji asynchronicznej niezbędne jest badanie stanu obiektu typu `XMLHttpRequest`. Odbywa się to przez sprawdzanie właściwości `readyState` oraz `status`. Pierwsza z nich odzwierciedla stan obiektu `XMLHttpRequest`. Zwykle przyjmuje się następujące wartości i stany:

- 0 — obiekt niezainicjowany (nie została wywołana metoda `open`, ang. *uninitialized*),
- 1 — dane obiektu w trakcie ładowania (została wywołana metoda `open`, ang. *loading*),
- 2 — dane obiektu zostały załadowane (została wywołana metoda `send`, ale odpowiedź serwera nie jest jeszcze dostępna, ang. *loaded*),
- 3 — część danych została odebrana i może zostać odczytana (ang. *interactive*),
- 4 — operacja zakończona (ang. *complete*).

Występują tu drobne różnice w stosunku do definicji rekomendowanej przez W3C, ale funkcjonalność pozostaje praktycznie taka sama. Definicja W3C jest następująca:

- 0 — stan `UNSENT`, obiekt utworzony, nie wykonywano na nim żadnych operacji;
- 1 — stan `OPENED`, została wywołana metoda `open`, obiekt gotowy do transmisji danych, można użyć metod `setRequestHeader` i `send`;
- 2 — stan `HEADERS_RECEIVED`, zostały otrzymane wszystkie nagłówki HTTP;
- 3 — stan `LOADING`, dane w trakcie pobierania;
- 4 — stan `DONE`, transmisja została zakończona.

Właściwość `status` pozwala na odczytanie kodu odpowiedzi, który zwrócił serwer w związku z żądaniem. Są to kody zgodne z protokołem HTTP i mogą przyjmować następujące wartości:

Kod	Nazwa	Znaczenie
1xx Kody informacyjne		
100	*Continue*	Serwer otrzymał nagłówki HTTP i oczekuje dalszych danych
101	*Switching Protocols*	Zmiana protokołu transmisji
2xx Kody potwierdzenia		
200	*OK*	Żądanie zostało prawidłowo przetworzone (zrealizowane)
201	*Created*	Żądanie spowodowało utworzenie nowego zasobu (dokumentu)
202	*Accepted*	Żądanie zostało zaakceptowane, ale nie zostało jeszcze przetworzone
203	*Non-Authoritative Information*	Żądanie nie jest autorytatywne, dane zostały pobrane z lokalnych lub zewnętrznych kopii
204	*No content*	Żądanie zostało wykonane, ale nie ma potrzeby zwracania żadnych danych
205	*Reset Content*	Żądanie zostało wykonane i klient powinien zresetować dokument (z reguły przywrócić jego pierwotną, poprzednią postać)
206	*Partial Content*	Serwer zrealizował tylko część żądania typu GET
3xx Kody przekierowania		
300	*Multiple Choices*	Istnieje kilka możliwości realizacji wskazanego żądania
301	*Moved Permanently*	Żądany zasób został na stałe przeniesiony pod inny adres
302	*Found*	Żądany zasób został tymczasowo przeniesiony pod inny adres
303	*See Other*	Odpowiedź na żądanie znajduje się pod innym (wskazanym) adresem
304	*Not Modified*	Zasób (dokument) nie zmienił się od ostatniego żądania i należy skorzystać z kopii lokalnej
305	*Use Proxy*	Żądany zasób jest dostępny tylko przez usługę proxy
306	-	Kod nieużywany w obecnych wersjach protokołu HTTP
307	*Temporary Redirect*	Żądany zasób jest tymczasowo dostępny pod innym adresem
4xx Kody błędów		
400	*Bad Request*	Nieprawidłowe zapytanie
401	*Unauthorized*	Żądanie wymaga prawidłowej autentykacji

Ciąg dalszy na stronie 2 ➔