улучшенная JavaScript

**Ненавязчивый JavaScript. Глава 1: Операция «Чистота» в действии**

Текст ниже — перевод статьи "Unobtrusive Javascript", чей автор, Christian Heilmann, любезно дал мне согласие на перевод. Оригинальный адрес статьи www.onlinetools.org/articles/unobtrusivejavascript/chapter1.html.

Веб-разработка в последние годы претерпела изменения: мы прекратили смешивать представление и структуру, таким образом менять дизайн и верстку по всему сайту стало легче — для этого нужно просто изменить таблицу стилей.

Дальнейшее разделение возможно, если использовать наследование и зависящие от контекста селекторы, вместо встроенных в элемент стилей и классов. (Прим. переводчика: статья о наследовании селекторов на русском.)

HTML:

<table border="0" cellpadding="0" width="100%" cellspacing="5">

<tr>

<td><font face="Arial" size="-2">Lorem Ipsum</font></td>

</tr>

</table>

превратился в

CSS:

td.content {font-family:Arial,sans-serif;font-size:12px;}

HTML:

<table cellpadding="0" style="width:100%;border:none" cellspacing="5">

<tr>

<td class="content">Lorem Ipsum</td>

</tr>

</table>

И в конечном итоге:

CSS:

body {font-family:Arial,sans-serif;font-size:12px;}

p {padding:5px;}

HTML:

<p>Lorem Ipsum</p>

Javascript может и должно ожидать такое же видоизменение.

Javascript: храним отдельно

Первое правило клуба ненавязчивого Javascript это никому не говорить о клубе ненавязчивого Javascript. Нет, нет, оставайтесь с нами, первое правило это:

1. Никогда, ни при каких обстоятельствах не добавляйте Javascript прямо в документ

Одна из сильных сторон Javascript, это то, что его можно поместить в отдельный файл. Так же, как и для CSS, это означает, что вы можете присвоить одну коллекцию функций для каждой страницы сайта, и если вам нужно изменить ее функциональность, вы можете сделать это в одном документе, что предпочтительнее, чем перебирать все страницы.

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

Это все что нам когда-либо понадобится, никакого больше встроенного Javascript. Смыть шампунь и повторить.

2. Javascript — это расширение, а не основная функциональность

Мы используем Javascript, только чтобы улучшить уже существующую функциональность, так как мы не можем полностью на него полагаться. Javascript может быть выключен или отфильтрован прокси-серверами и файерволами компаний обеспечивающих безопасность. Мы никогда не можем принимать Javascript за данноcть.

Это не означает, что мы совсем не можем использовать Javascript, это только значит, что мы можем добавлять его лишь как дополнительную возможность.

HTML:

<form action="index.php" onsubmit="return checkform(this)">

<p><label for="login">Login:</label>

<input type="text" name="login" id="login" /></p>

<p><label for="pw">Password:</label>

<input type="password" name="pw" id="pw" /></p>

<p><input type="submit" value="send" /></p>

</form>

Javascript:

function checkform(f)

{

var error='';

error+=f.login.value==''?'\nlogin':'';

error+=f.pw.value==''?'\npassword':'';

if (error!='')

{

alert('Please enter the following:'+error);

}

return error=='';

}

Код совершенно правильный и ограничит пользователей от попыток отправить пустую форму.

HTML:

<form action="index.php">

<p><label for="login">Login:</label>

<input type="text" name="login" id="login" /></p>

<p><label for="pw">Password:</label>

<input type="password" name="pw" id="pw" /></p>

<p><input type="button" onclick="checkform()" value="send" /></p>

</form>

Javascript:

function checkform()

{

var f=document.forms[0];

var error='';

error+=f.login.value==''?'\nlogin':'';

error+=f.pw.value==''?'\npassword':'';

if (error!='')

{

alert('Please enter the following:'+error);

} else {

f.submit();

}

}

Вроде бы то же самое, но есть одна существенная проблема: если Javascript выключен, кнопка ничего не делает, и не важно, как часто расстроенный пользователь кликает по ней.

Давайте повторим: Javascript не надежен, не будем полагаться на него!

3. Проверяйте доступность объекта прежде чем использовать его

Множество ошибок Javascript происходит просто потому, что программист был слишком ленив, чтобы проверить, доступен или нет данный объект или метод.

Javascript:

function color(o,col)

{

o.style.background=col;

}

может окончиться ошибкой Javascript, если объект о не доступен.

Javascript:

function color(o,col)

{

if(o)

{

o.style.background=col;

}

}

Данный пример работает всегда.

Этот прием используется, в основном, чтобы проверить способность браузера поддерживать конкретную функциональность Javascript. Во времена проб и ошибок в программировании (не важно кем оплаченных, клиентами или компьютерной фирмой на которую мы работаем) это достигалось с помощью скрипта анализирующего тип браузера, что есть концепция нежизнеспособная с самого начала (каждый раз когда браузер обновляется или появляется новый, этот скрипт нужно обновлять). В большинстве примеров используемых в наши дни, необходимо проверить, способен ли браузер понимать W3C DOM, что дает нам следующие правило:

4. Не создавайте Javascript на браузеро-зависимых диалектах

Если нет действительно весомой причины, мы никогда не должны использовать специфичные для браузера расширения веб-стандартов. Времена проверки document.layers (Netscape 4.x) или document.all (Internet Explorer < 5) прошли. Все современные браузеры поддерживают свойство DOM document.getElementsById и именно это мы используем для проверки.

Javascript:

function doDOMstuff()

{

if(document.getElementById && document.createTextNode)

{

[...]

}

}

Вторая проверка необходима только для некоторых старых версий Opera, которые утверждали, что понимают DOM, но делали это неверно.

5. Не похищайте чужие переменные скрипта

Когда мы создаем функцию или функциональность (functionality), мы должны убедиться, что все использованные переменные локальны, чтобы избежать переписывания одной функцией переменных, которые используются в другой.

Javascript:

var i=42; // глобальная переменная

function dothings()

{

for (i=0;i<200;i++) // i изменяется

{

// какой-то код

}

}

function dothingsright()

{

var i; // определяем i локально

for (i=0;i<200;i++)

{

// какой-то код

}

}

alert(i); // = 42

dothingsright()

alert(i) // = 42 (обусловлено локальным определением)

dothings()

alert(i) // = 200, ой-ой!

6. Сохраняйте функциональность независимой от наличия мыши

Мало убедиться, что все работает при подключенном Javascript. Мы также должны принимать во внимание пользователей, которые могут не использовать мышь вообще или испытывают с этим трудности. Следовательно, мы должны гарантировать, что все эффекты запускаются и при помощи мыши, и с клавиатуры.

Самая большая проблема с независимостью от наличия мыши — это поля формы, чьи значения мы проверяем, или которые запускают Обработчики события onchange или onblur. Не используйте их, это так просто (as simple as that).

HTML:

<form>

<p>

<label for="go2">Go to:</label>

<select name="go2" id="go2" onchange="send(this)">

<option value="index.html">Home</option>

<option value="chapter1.html">Operation Cleanout</option>

<option value="chapter2.html">Reaching things</option>

</select>

</p>

</form>

Javascript:

function send(f)

{

var chosen;

chosen=f.options[f.selectedIndex].value;

self.location=chosen;

}

Проверьте этот пример недоступного выпадающего списка

Видимо, главная проблема в том, что если вы переходите к элементу, используя клавиатуру, и нажимаете стрелку вниз, то для того чтобы выбрать что-либо, вы никогда не пройдете мимо первого пункта. Ведь send() запускается каждый раз, когда вы переходите с первого на второй пункт.

Хотя опытные клавиатурщики могут знать, что если нажать alt+ стрелку вниз, выпадающее меню полностью развернется, и можно будет выбрать нужный пункт.

HTML:

<form action="send.php" method="post" onsubmit="return send(this)">

<p>

<label for="go2">Go to:</label>

<select name="go2" id="go2">

<option value="index.html">Home</option>

<option value="chapter1.html">Operation Cleanout</option>

<option value="chapter2.html">Reaching things</option>

</select>

<input type="submit" value="go" />

</p>

</form>

Javascript:

function send(f)

{

var chosen;

chosen=f.go2.options[f.go2.selectedIndex].value;

self.location=chosen;

return false;

}

PHP:

<?PHP if(isset($\_POST['go2?])){

header('Location:'.$\_POST['go2?]);

}?>

Проверить этот пример доступного выпадающего списка

Это работает сравнительно хорошо и экономит нам один запрос сервера к перенаправляющему скрипту send.php, который отрабатывает, только когда не доступен Javascript.

Как насчет onkeypress?

Когда мы читаем руководства по доступному веб-контенту, везде рекомендуют использовать независимые от устройств обработчики событий в наших скриптах.

«По-другому, если вы должны использовать независимые от устройств атрибуты, обеспечьте избыточный механизм ввода (то есть определите два обработчика для одного и того же элемента):»

Используйте onmousedown вместе с onkeydown.

Используйте onmouseup вместе с onkeyup

Используйте onclick вместе с onkeypress

Это звучит великолепно в теории, но в реальных жизненных ситуациях обрабочик onkeypress плохо поддерживается различными браузерами. Пользователям, зависящим от просмотра с клавиатуры, для имитации клика обычно дается либо клавиша Enter, либо пробел, либо они обе. Используя onkeypress, мы можем украсть другую клавиатурную функциональность нужную пользователю. Примеры опережающий ввод в Мозилле, дополнительные горячие клавиши в Опере (типа A для перехода к следующей ссылке) или клавиатурные средства управления в JAWS.

Ссылки:

The World Wide Web Consortium: www.w3.org

The w3c Document Object Model: www.w3.org/DOM/

Web Content Accessibility Guidelines: www.w3.org/TR/WCAG10-HTML-TECHS/#directly-accessible-scripts

Mozilla's type-ahead: www.mozilla.org/projects/ui/accessibility/typeaheadfind.html

Opera's keyboard controls: www.opera.com/features/keyboard/complete/

JAWS keyboard controls: www.webaim.org/techniques/jaws/keyboard-shortcuts

Читать дальше

Ненавязчивый JavaScript. Глава 2: Как достичь того, что мы хотим изменить

Ненавязчивый JavaScript. Глава 3: Создание и уничтожение элементов страницы

Ненавязчивый JavaScript. Глава 4: Вызываем ветер (скрипты)

Ненавязчивый JavaScript. Глава 5: Пример форма, улучшенная JavaScript

**Ненавязчивый JavaScript. Глава 2: Как достичь того, что мы хотим изменить**

Текст ниже — перевод статьи "Unobtrusive Javascript", чей автор Christian Heilmann любезно дал мне согласие на перевод. Оригинальный адрес статьи www.onlinetools.org/articles/unobtrusivejavascript/chapter2.html

Для неопытных Javascript-разработчиков HTML — это площадка для игр.

HTML:

<a href="index.html"

onmouseover="image1.src='1on.gif'"

onmouseout="image1.src='1off.gif'">

<img src="1off.gif" name="image1" border="0" height="150" width="150"

alt="home"></a>

или если они чуть более продвинутые:

HTML:

<a href="index.html"

onmouseover="roll('home',1)"

onmouseout="roll('home',0)">

<img src="home.gif" name="home" border="0"

height="150" width="150"

alt="home"></a>

Javascript:

// предзагрузка изображений

homeoff = new Image();

homeoff.src = 'home.gif';

homeon = new Image();

homeon.src = 'homeoff.gif';

function roll(imgName,a)

{

imgState=a==0?eval(imgName + 'on.src'):eval(imgName + 'off.src');

document.images[imgName].src = imgState;

}

В любом случае все вызовы событий находятся в HTML, и если имя функции поменяется, нам нужно будет внести изменения в каждом документе. Кроме того, каждый ролловер будет означать большее количество разметки, которая добавиться к общему весу страницы.

Изыди! Изыди! — вы одержимы встроенными в документ вызовами событий.

Давайте забудем на мгновение, что почти каждый эффект ролловера в наши дни скорее достигается с помощью CSS, чем с помощью Javascript, и допустим, что мы хотим использовать нижеследующую разметку и создать ролловер для изображения.

HTML:

<a href="index.html"><img src="home.gif" id="home" alt="home"></a>

Итак, как сменить изображение, когда на него наводиться курсор мыши?

Карабкаемся по ветвям дерева узлов (node tree)

Каждый XML- (а также и HTML-) документ — это дерево узлов. Узел — часть этого дерева (представляйте файл или папку в проводнике Windows, когда вы просматриваете ваш диск). Узел может быть двенадцатью разными вещами, в HTML только три из них действительно интересны: element, TextNode и AttributeNode.

Наше альпинистское снаряжение.

Давайте посмотрим, какие функции и атрибуты мы можем использовать, чтобы перемещаться по дереву узлов документа, и как перепрыгивать от одного элемента до другого.

Функции для достижения элемента страницы

getElementById('elementID')

возвращает элемент с id elementID в виде объекта.

getElementsByTagName('tag')

возвращает все элементы с именем tag в виде массива.

Разумеется, мы можем смешивать и сочетать эти два варианта. Несколько примеров:

document.getElementById('navigation').getElementsByTagName('a')[3];

// возвращает четвёртую ссылку внутри элемента,

// который имеет ID 'navigation'

document.getElementsByTagName('div')[2].getElementsByTagName('p')[0];

// возвращает первый параграф внутри третьего div в документе.

Инструменты, чтобы переместиться к конкретному элементу

childNodes

возвращает массив узлов, лежащих внутри текущего. Существуют также firstChild и lastChild, это сокращенные варианты childNodes[0] и childNodes[this.childNodes.length-1].

parentNode

элемент, содержащий данный

nextSibling

следующий элемент того же уровня в дереве документа

previousSibling

предыдущий элемент того же уровня в дереве документа

Все это по желанию может быть смешано так, как нам нужно.

Javascript:

var other=document.getElementById('nav').childNodes[3].firstChild;

// возвращает четвёртый элемент первого подэлемента

// внутри элемента с ID=nav

var prevlink=o.parentNode.previousSibling.firstChild.childnodes[2];

// возвращает третий узел внутри предыдущего элемента,

// того же уровня, что и родительский элемент элемента о.

Атрибуты и функции элементов

attributes

возвращает массив атрибутов этого элемента. Не работает с IE версией менее 6.

data

возвращает или устанавливает текстовые данные узла

nodeName

возвращает имя узла (имя HTML элемента)

nodeType

возвращает тип узла: 1, если элемент — это узел, 2 — если атрибут, 3 — если текст.

nodeValue

возвращает или устанавливает значение узла. Это значение является текстом, если узел — текстовый, атрибутом, если это узел-атрибут, или null, если это элемент.

getAttribute (attribute)

возвращает значение атрибута attribute.

Javascript:

var other=document.getElementById('nav').firstChild;

if(other.nodeType==3)

{

other.data='newtext';

}

if(other.nodeType==1)

{

other.firstChild.data='newtext';

}

Итак, чтобы достичь изображения в нашем примере, мы будем использовать либо getElementsByTagName, либо getElementById.

HTML:

<a href="index.html"><img src="home.gif" id="home" alt="home"></a>

Javascript:

function findimg()

{

var image;

image=document.getElementById('home');

if (image)

{

image.style.border='3px dashed #ccc';

}

}

//или:

function findimg()

{

var imgs,i;

imgs=document.getElementsByTagName('img');

for(i in imgs)

{

if(/home.gif/.test(imgs[i].src))

{

imgs[i].style.border='3px dashed #ccc';

}

}

}

Использовать getElementById гораздо проще, так как нам не нужно пробегать все элементы и искать уникальный идентификатор. В этом примере мы проверяем, содержит ли атрибут src объекта изображения значение 'home.gif'. Более общий способ — проверить назначен ли элементу специальный класс.

HTML:

<a href="index.html"><img src="home.gif" class="roll" alt="home"></a>

Javascript:

function findimg()

{

var imgs,i;

imgs=document.getElementsByTagName('img');

for(i in imgs)

{

if(/roll/.test(imgs[i].className))

{

imgs[i].style.border='3px dashed #ccc';

}

}

}

Итак, чтобы добавить эффект ролловера, все что нам надо сделать — это добавить функцию, которая переключает источник изображения и прикрепляет обработчик события к картинке.

function findimg()

{

var imgs,i;

// перебираем все изображения в документе

imgs=document.getElementsByTagName('img');

for(i=0;i<imgs.length;i++)

{

// проверяем существует ли класс 'roll'

if(/roll/.test(imgs[i].className))

{

// добавляем функцию roll на события onmouseover и onmouseout,

//затем посылаем само изображение как объект

imgs[i].onmouseover=function(){roll(this);};

imgs[i].onmouseout=function(){roll(this);};

}

}

}

function roll(o)

{

var src,ftype,newsrc;

// послать src изображения и найти расширение файла

src = o.src;

ftype = src.substring(src.lastIndexOf('.'), src.length);

// проверить содержит ли источник изображения строку \_on и,

// если да - удалить её

if(/\_on/.test(src))

{

newsrc = src.replace('\_on','');

}else{

// в противном случае добавить \_on к src

newsrc = src.replace(ftype, '\_on'+ftype);

}

o.src=newsrc;

}

window.onload=function(){

findimg();

}

Здесь можно проверить этот пример

Вроде бы все хорошо, но мы забыли одну вещь: хотя то, что получилось, просто замечательно, ролловер должен работать и без мыши. Чтобы достигнуть этого, мы должны проверить, находиться ли фокус на ссылке вокруг изображения, так как сама картинка, если на неё поставлена ссылка, недостижима с клавиатуры.

Чтобы сделать проверку, нам надо получить элемент, который содержит изображение, в данном случае — ссылку. Мы сделаем это с помощью команды parentNode. Так как это заодно изменяет объект, которые передан как параметр функции roll(), нам надо будет найти изображение снова. Поэтому мы перебираем дочерние узлы ссылок и смотрим, который из них одновременно и элемент, и изображение, проверив nodeType and nodeName. Это необходимо, так как некоторые браузеры рассматривают пробелы в источнике изображения как узел, другие же нет.

function findimg()

{

var imgs,i;

// перебираем все изображения, проверяем содержат ли они класс roll

imgs=document.getElementsByTagName('img');

for(i=0;i<imgs.length;i++)

{

if(/roll/.test(imgs[i].className))

{

// добавляем функцию roll к родительскому элементу изображения

imgs[i].parentNode.onmouseover=function(){roll(this);};

imgs[i].parentNode.onmouseout=function(){roll(this);};

imgs[i].parentNode.onfocus=function(){roll(this);};

imgs[i].parentNode.onblur=function(){roll(this);};

}

}

}

function roll(o)

{

var i,isnode,src,ftype,newsrc,nownode;

// перебираем все дочерние узлы

for (i=0;i<o.childNodes.length;i++)

{

nownode=o.childNodes[i];

// если узел это и элемент, и IMG,

// то установить значение переменной и выйти из цикла

if(nownode.nodeType==1 && /img/i.test(nownode.nodeName))

{

isnode=i;

break;

}

}

// проверить src и сделать ролловер

src = o.childNodes[isnode].src;

ftype = src.substring(src.lastIndexOf('.'), src.length);

if(/\_on/.test(src))

{

newsrc = src.replace('\_on','');

}else{

newsrc = src.replace(ftype, '\_on'+ftype);

}

o.childNodes[isnode].src=newsrc;

}

window.onload=function(){

findimg();

}

Проверьте этот независимый от наличия мыши код

Почему бы вам не попробовать?

Просто скачайте демонстрационный HTML и попробуйте сделать одно из следующих заданий. Пройдите по ссылке рядом с заданием, чтобы увидеть одно из возможных решений. Решения, в качестве демонстрации, имеют встроенный Javascript, и код не выделяется в отдельный документ так, как это положено. Это сделано, чтобы помочь вам увидеть всю необходимую разметку в одном документе, это легче чем в двух разных.

Поменяйте цвет каждого H2 на голубой. Решение для смены цвета.

Обведите каждый второй параграф чёрной рамкой. Решение для обведённых параграфов

Проверьте каждую ссылку в документе, является ли она внешней (проверяя, что атрибут href не содержит window.location.hostname) и добавьте href каждой ссылки в круглых скобках после ссылки. Пока вы ещё только изучаете, как это сделать правильно, используйте атрибут innerHTML объекта ссылки, чтобы изменить его содержание. Решение для внешних ссылок.

Добавьте обработчик события onclick, который открывает всплывающее окно 400x400, к каждой ссылке с атрибутом target. Решение для всплывающих окон

Читать дальше

Ненавязчивый JavaScript. Глава 1: Операция «Чистота» в действии

Ненавязчивый JavaScript. Глава 3: Создание и уничтожение элементов страницы

Ненавязчивый JavaScript. Глава 4: Вызываем ветер (скрипты)

Ненавязчивый JavaScript. Глава 5: Пример форма, улучшенная JavaScript

© 2016 ООО ВЕБО

E-mail:

info@webo.name

Блог:

blog.webogroup.com

Twitter:

wboptimizer

Продукты

Облачное ускорение сайтов

WEBO Site SpeedUp

WEBO Pulsar

WEBO Enterprise SpeedoMeter

Информация

О компании

Вакансии

Контакты

Карта сайта

О проекте

Проверка скорости

Статьи

РегистрацияЛичный кабинет

Ускорение вашего сайта. Гарантированный результат.

Проверка скорости сайта

Ускорение сайта «под ключ»

WEBO Site SpeedUp

Книги по оптимизации

«Разгони свой сайт»

«Реактивные веб-сайты»

Yet Another cSS selector

Самая быстрая библиотека для выбора элементов по CSS-селектору

Ссылки по теме

Web Optimizer для автоматизации клиентской оптимизации

Способы ускорения загрузки Вашего сайта

Анализируем загрузку веб-страницы

Оптимизируем загрузку веб-страницы

Техника CSS sprites

CSS Sprites: все, что вы знали, но боялись спросить

Auto CSS Sprites

Картинки в теле страницы с помощью data:URL

DURIS.ru — data:URL Sprites

Приемы для JavaScript

javascript.ru — самоучитель, форум, информация

Высокопроизводительные AJAX-приложения

Javascript: жать или не жать?

Практический JS: «отложенная» загрузка

Хитрости для CSS

CSS: все о сжатии

Практический CSS/JS: архивируем все!

Разгоняем CSS-селекторы: стоит ли?

Облако сайтов

Автор: Додонов Александр

Опубликована: 18 января 2006, aboutweb2.spb.ru/nenavyazchivy-javascript/glava-3-sozdanie-i-unichtozhenie-elementov-stranitsyi/

Ненавязчивый JavaScript. Глава 3: Создание и уничтожение элементов страницы

Текст ниже — перевод статьи "Unobtrusive Javascript", чей автор Christian Heilmann любезно дал мне согласие на перевод. Оригинальный адрес статьи www.onlinetools.org/articles/unobtrusivejavascript/chapter3.html

Основное достоинство DOM это возможность не только читать, но также изменять содержание и структуру находящегося в наличии документа. Для этого в нашем распоряжении есть несколько методов

Создание нового содержания

createElement(element)

Создание нового элемента

createTextNode(string))

Создание нового текстового узла со значением string.

Только что созданные элементы не добавляются в документ немедленно, они пребывают в чистилище до тех пор, пока мы не добавим их куда-нибудь в дерево узлов.

Javascript:

mynewparagraph=document.createElement('p');

mynewtext=document.createTextNode('this is a new paragraph');

Изменение существующего содержания

setAttribute(attribute,value)

Добавляет новый атрибут с соответствующим значением к объекту

appendChild(child)

Добавляет дочерний узел к объекту, этот узел должен быть объектом, строку использовать нельзя.

cloneNode()

Копирует узел со всеми дочерними узлами.

hasChildNodes()

Проверяет, имеет ли объект дочерние узлы и если да, возвращает true

insertBefore(newchild,oldchild)

Добавляет новый узел newchild перед узлом oldchild в дереве документа.

removeChild(oldchild)

Удаляет дочерний узел oldchild.

replaceChild(newchild,oldchild)

Перемещает узел newchild на место узла oldchild

removeAttribute(attribute)

Удаляет атрибут attribute объекта.

Пример с картинкой

Пусть у нас есть ссылки на изображения, и они должны открываться в новом окне браузера без Javascript, или под ссылками, если Javascript подключен.

HTML:

<ul id="imglist">

<li><a href="home.gif" target="\_blank">Home

(new window)</a></li>

<li><a href="home\_on.gif" target="\_blank">Home active

(new window)</a></li>

<li><a href="jscsshtml.gif" target="\_blank">HTML-CSS-Javascript

(new window)</a></li>

</ul>

Итак, в случае, когда Javascript и DOM доступны, мы хотим:

Удалить надпись «(new window)» из ссылки

Добавить обработчик события для вызова функции popw()

Эта функция должна:

Показывать изображение, соответствующее ссылке, под ссылкой, если его еще нет.

Удалять изображение, если оно уже есть (чтобы не добавлять снова и снова изображение, нажимая по ссылке)

Сделать, чтобы изображение исчезало, когда пользователи кликают по нему.

Первую задачу решить нетрудно:

Javascript:

function imgpop()

{

var il,imga,imgatxt;

// получаем все LI из списка изображений, пробежаться по ним циклом

il=document.getElementById('imglist').getElementsByTagName('li');

for(i=0;i<il.length;i++)

{

// вычленяем первую ссылку из LI

imga=il[i].getElementsByTagName('a')[0];

// удаляем выражение (new window) из текста ссылки

// (который является содержанием (nodeValue) первого узла)

imgatxt=imga.firstChild;

imgatxt.nodeValue=imgatxt.nodeValue.replace(/ \(new window\)/,'');

// добавляем обработчик события для вызова popw();

imga.onclick=function(){return popw(this);}

//imga.onkeypress=function(){return popw(this);}

}

}

Далее, для функции popw() мы должны использовать некоторые из методов описанных выше:

Javascript:

function popw(o)

{

var newimg;

// если уже есть изображение в parentNode (li)

if(o.parentNode.getElementsByTagName('img').length>0)

{

// удаляем его

o.parentNode.removeChild(o.parentNode.getElementsByTagName('img')[0]);

} else {

// иначе создаем новое изображение и добавляем обработчик,

// который удаляет изображение, когда вы кликаете по нему

newimg=document.createElement('img');

newimg.style.display='block';

newimg.onclick=function(){this.parentNode.removeChild(this);};

newimg.src=o.href;

o.parentNode.appendChild(newimg)

}

return false;

}

Посмотреть работу скрипта показа изображений

Пример: ссылка на указатель даты

Пусть, например, у нас есть форма, содержащая поля даты и мы хотим предложить указатель даты пользователям, у которых подключен Javascript, и вводить дату вручную — прочим пользователям. Давайте не будем обсуждать здесь саму реализацию указателя даты, а сфокусируемся на том, как его вызвать.

Сначала напишем необходимый HTML. Чтобы видеть какие элементы должны выдавать ссылку на указатель даты, мы добавим к ним классы date.

HTML:

<h1>Flight booking</h1>

<form action="nosend.php" method="post" onsubmit="return check(this);">

<p>Step 1 of 4</p>

<h2>Please select your dates</h2>

<p>

<label for="startdate">Start Date</label>

<input type="text" class="date" id="startdate" name="startdate" />

</p>

<p>

<label for="enddate34;>End Date</label>

<input type="text" class="date" id="enddate" name="enddate" />

</p>

<p>

<input type="submit" value="send" />

</p>

</form>

Мы перебираем все input'ы документа, и проверяем, у какого из них содержится класс date в className (помните, элементы могут иметь более чем один класс в атрибуте class!)

Когда мы находим этот класс, то создаем новый объект ссылки и ее текст. Этот текст мы присоединяем к объекту ссылки как дочерний объект и добавляем обработчик события, чтобы вызвать наш скрипт указателя даты.

Как только ссылка создана, мы добавляем ее после поля ввода.

Javascript:

function addPickerLink()

{

var inputs,pickLink,pickText;

// перебираем все input'ы

inputs=document.getElementsByTagName('input');

for(i=0;i<inputs.length;i++)

{

// если class содержит ‘date'

if(/date/.test(inputs[i].className))

{

// создать новую ссылку с текстом

pickLink=document.createElement('a');

pickText=document.createTextNode('pick a date');

// добавляем текст дочерним узлом к ссылке

pickLink.appendChild(pickText);

// устанавливаем атрибут href в #

// и вызываем picker на клик или переход с клавиатуры

pickLink.setAttribute('href','#');

pickLink.onclick=function(){picker(this);return false;};

//pickLink.onkeypress=function(){picker(this);return false;};

// добавляем новую ссылку к родителю поля ввода (это P)

inputs[i].parentNode.appendChild(pickLink)

}

}

}

Посмотреть демонстрационный пример

Сейчас после всех полей с датами есть ссылка на picker().

Все что нам надо, это сказать функции picker к чему употребить возвращаемое значение.

Ссылка передает сама себя функции picker() в виде объекта, и мы должны получить доступ к предыдущему узлу того же уровня — INPUT.

Javascript:

function picker(o)

{

alert('This is a simulation only.') // это демонстрационная функция

o.previousSibling.value='26/04/1975?;

}

Готово, но не вполне. Мы добавили новую ссылку как последнего ребенка к родительскому узлу input, а ведь очень даже может быть, что предыдущий узел того же уровня, что и наша ссылка на самом деле не INPUT, а пробел! Следовательно, нам надо перебирать все предыдущие узлы того же уровня пока мы не наткнемся на узел типа элемент.

Javascript:

function picker(o)

{

alert('This is a simulation only.') // это демонстрационная функция

while(o.previousSibling.nodeType!=1)

{

o=o.previousSibling;

}

o.previousSibling.value='26/04/1975?;

}

Перебор всегда операция нудная (hacky) и зачастую довольно медленная. Чтобы избежать перебора мы должны изменить нашу функцию.

Изменение функции addPickerLink().

Использовать addPickerLink() легко, но это делает нас зависимым от разметки. Что случится если, к примеру, нам нужно будет добавить впоследствии за тэгом input — тэг SPAN со звездочкой (\*), чтобы указать, что это поле необходимое?

Обходной маневр — это использовать insertBefore() для следующего узла того же уровня, что и наше поле ввода.

Javascript:

function addPickerLink()

{

var inputs,pickLink,pickText;

// перебираем все input'ы

inputs=document.getElementsByTagName('input');

for(i=0;i<inputs.length;i++)

{

// если class содержит 'date'

if(/date/.test(inputs[i].className))

{

// создаем новый элемент ссылки и текст

pickLink=document.createElement('a');

pickText=document.createTextNode('pick a date');

// добавляем текст как дочерний узел к ссылке

pickLink.appendChild(pickText);

// устанавливаем атрибут href равным #

// и вызываем picker при клике или переходе с клавиатуры

pickLink.setAttribute('href','#');

pickLink.onclick=function(){picker(this)};

//pickLink.onkeypress=function(){picker(this)};

// добавляем новую ссылку сразу после input'а

inputs[i].parentNode.appendChild(pickLink)

inputs[i].parentNode.insertBefore(pickLink,inputs[i].nextSibling);

}

}

}

Посмотреть демонстрационный пример.

Памятки

Ну, вот и все, с описанными инструментами мы способны получить доступ к любому элементу документа и затем изменить этот элемент, и мы можем улучшить работу пользователя, не привязываясь к Javascript.

Возможно, поначалу применять DOM покажется немного трудным, но как только вы настроите ваше мышление для использования DOM, вам с каждым разом будет легче его применять.

Некоторые проблемы общего характера:

Убедитесь что элемент существует, прежде чем попытаться обратиться к нему. Некоторые браузеры счастливы проверить object.nextSibling.nodeName и возвращают false когда следующего узла нет или же этот узел текстовый, но другие генерируют ошибку, когда вы пытаетесь обратиться к атрибуту несуществующего элемента.

Убедитесь, что код не привязан слишком сильно к верстке, перевод строки могут быть прочитаны как новый узел, разметка может поменяться, а ведь вам неохота менять ваш скрипт с каждой сменой дизайна.

Чтение содержания элемента осуществляется чтением значения его childNodes, а не самого элемента! document.getElementsByTagName('h2')[0].nodeValue пуст, document.getElementsByTagName('h2')[0].firstChild.nodeValue — нет.

Проверяя nodeNames и атрибуты, убедитесь, что вы указываете правильный регистр , так как одни браузеры считают, что элементы пишутся в вернем регистре, другие же, что в нижнем.

HTML сгенерированный DOM в большинстве случаев синтаксически неверен, если вы хотите повторно использовать HTML из сгенерированной браузером странице, вы должны вычистить его.

Избегайте частого перебора элементов, если вы можете сами создать разметку, с которой вам предстоит работать, стремитесь использовать ID вместо перебора.

Знайте ваш синтаксис. Многочисленное (Many a time) использование getElementsById может послужить причиной частого переписывания скрипта (rescripting).

Знайте ваши объекты Javascript и HTML атрибуты. Это означает, что не нужно проверять атрибут, которого нет.

Не рассчитывайте, что на других сайтах придерживаются ваших приемов верстки. Например, стоит проверять, что className содержит вашу строку, а не то, что он ей равен, так как некоторым программистам нравиться применять несколько классов к одному элементу.

Как насчет innerHTML?

Когда появился Internet Explorer 4, родился innerHTML — быстрый способ создания и изменения контента. Этот способ прочитать содержание элементов много проще чем рекомендованный W3C, главным образом, в случае, если элемент содержит дочерние узлы, которые сами элементы, а нам надо прочитать содержание целиком. Чтобы сделать это с помощью одного только DOM, вам придется пройти сквозь мучительные испытания — проверять тип и читать значения каждого узла.

Свойство innerHTML много проще использовать, но у него есть несколько недостатков. Например, вы не можете получить обратно ссылки на элементы, созданные с его помощью, так как итоговое значение скорее строка, чем объект. Кроме того, innerHTML поддерживается только HTML, но не XML, в отличие от DOM, которая обеспечивает переносимость на любую разметку. Просмотрите раздел DOM в Quirksmode.org или всестороннее исследование в Developer-x, чтобы сравнить поддержку DOM разными браузерами.

Читать дальше

Ненавязчивый JavaScript. Глава 1: Операция «Чистота» в действии

Ненавязчивый JavaScript. Глава 2: Как достичь того, что мы хотим изменить

Ненавязчивый JavaScript. Глава 4: Вызываем ветер (скрипты)

Ненавязчивый JavaScript. Глава 5: Пример форма, улучшенная JavaScript

© 2016 ООО ВЕБО

E-mail:

info@webo.name

Блог:

blog.webogroup.com

Twitter:

wboptimizer

Продукты

Облачное ускорение сайтов

WEBO Site SpeedUp

WEBO Pulsar

WEBO Enterprise SpeedoMeter

Информация

О компании

Вакансии

Контакты

Карта сайта

О проекте

Проверка скорости

Статьи

РегистрацияЛичный кабинет

Ускорение вашего сайта. Гарантированный результат.

Проверка скорости сайта

Ускорение сайта «под ключ»

WEBO Site SpeedUp

Книги по оптимизации

«Разгони свой сайт»

«Реактивные веб-сайты»

Yet Another cSS selector

Самая быстрая библиотека для выбора элементов по CSS-селектору

Ссылки по теме

Web Optimizer для автоматизации клиентской оптимизации

Способы ускорения загрузки Вашего сайта

Анализируем загрузку веб-страницы

Оптимизируем загрузку веб-страницы

Техника CSS sprites

CSS Sprites: все, что вы знали, но боялись спросить

Auto CSS Sprites

Картинки в теле страницы с помощью data:URL

DURIS.ru — data:URL Sprites

Приемы для JavaScript

javascript.ru — самоучитель, форум, информация

Высокопроизводительные AJAX-приложения

Javascript: жать или не жать?

Практический JS: «отложенная» загрузка

Хитрости для CSS

CSS: все о сжатии

Практический CSS/JS: архивируем все!

Разгоняем CSS-селекторы: стоит ли?

Облако сайтов

Автор: Додонов Александр

Опубликована: 18 января 2006, aboutweb2.spb.ru/nenavyazchivy-javascript/glava-4-vyizyivaem-veter-skriptyi/

Ненавязчивый JavaScript. Глава 4: Вызываем ветер (скрипты)

Текст ниже — перевод статьи "Unobtrusive Javascript", чей автор Christian Heilmann любезно дал мне согласие на перевод. Оригинальный адрес статьи http://www.onlinetools.org/articles/unobtrusivejavascript/chapter4.html

Главная техника, которую мы используем, чтобы сохранить наш Javascript ненавязчивым, это хранение скрипта в отдельном файле, что предпочтительней, чем смешивать его с разметкой страницы. Чтобы исполнить функции в нашем .js файле, нам надо вызвать их, когда страница загружена. Это может быть достигнуто различными способами, каждый имеет свои достоинства и недостатки.

Старая школа

Когда-то давно, когда мы были молоды и невинны, мы добавляли атрибут onload к элементу body.

HTML:

<body onload="foo();">

Что нам надо запомнить — мы поступали плохо и больше так не будем!

Если мы вызываем скрипт(ы) в элементе body, мы возвращаемся к старому, так как продолжаем смешивать разметку и вызовы событий. Переместить вызов скрипта в файл .js — вот что нам надо сделать.

Способ с отделением

Мы вызываем скрипт, прикрепляя вызов к событию onload объекта window.

Когда у нас только одна функция, мы не ставим круглых скобок на конце имени функции, это должно возвращать результаты функции лучше, чем запускать функцию.

Если у нас более чем одна функция, мы должны, чтобы вызвать остальные, использовать анонимную функцию, в этот раз уже с круглыми скобками.

Javascript:

window.onload=foo;

или

window.onload=function(){

foo();

bar();

baz();

}

Этот метод вызова функций применим, конечно, не только к объекту window. Как показано ранее, мы можем добавить его к любому объекту страницы. Это поддерживается каждым Javascript/DOM браузером.

Недостаток этого решения в том, что мы не совсем ненавязчивы. Если у нас больше чем один подключенный .js, вызываемые скрипты могут переписать вызовы onload один у другого.

По-настоящему ненавязчивый способ

Существует возможность добавлять обработчики событий в комплект к уже существующим обработчикам. Довольно печально, что различные браузеры оснащены этой функциональностью по-разному, а IE на Mac'ах так вообще не оснащен.

Эта функциональность реализована, например, в прекрасной многократно используемой функции, написанной Scott Andrew. При вызове функции вы передаете ей объект, который нужно привязать к событию, тип события и имя функции.

function addEvent(obj, evType, fn){

if (obj.addEventListener){

obj.addEventListener(evType, fn, false);

return true;

} else if (obj.attachEvent){

var r = obj.attachEvent(\"on\"+evType, fn);

return r;

} else {

return false;

}

}

addEvent(window, 'load', foo);

addEvent(window, 'load', bar);

Очевидный существующий недостаток в том, что функция не привязывает ничего в IE на Macintosh.

Голос Оперы

Я получил письмо от службы технической поддержки Opera объясняющее, что в предыдущем примере есть упущения, поэтому теперь он отличается от примера на сайте Скотта.

«Функция addEvent не вполне соответствует стандартам. Она вроде бы должна работать в FireFox, но с тех пор как они пофиксили этот баг, она наверняка не будет больше работать в FireFox. Функция возможно не будет делать того что вы хотите в Опере, которая поддерживает стандарты правильно.»

«Проблема в том, что вы настраиваете "перехватывающий" обработчик события, используя true в качестве третьего аргумента в функции addEventListener:»

if (obj.addEventListener){

obj.addEventListener(evType, fn, true);

«Этому true следовало быть false. Если переменная равна true, прослушка события не должна возбуждаться ни для одного элемента для которого она зарегистрирована. Например, если вы используете это с элементом LI, она не будет ничего делать пока LI имеет дочерние элементы. Проверьте в Опере :)»

И еще немного

События — очень многоплановая и запутанная тема в Javascript. Для разработки веб-сайтов, указанных примеров достаточно, но если мы переходим к разработке веб-приложений, мы можем захотеть большего. Проверьте указанные в дополнении ссылки к полезным статьям, чтобы получить больше информации о событиях и о том, как привязывать их к элементам.

Ссылки

[1] Функция, которая добавляет дополнительные обработчики к событию. www.scottandrew.com/weblog/articles/cbs-events

Читать дальше

Ненавязчивый JavaScript. Глава 1: Операция «Чистота» в действии

Ненавязчивый JavaScript. Глава 2: Как достичь того, что мы хотим изменить

Ненавязчивый JavaScript. Глава 3: Создание и уничтожение элементов страницы

Ненавязчивый JavaScript. Глава 5: Пример форма, улучшенная JavaScript

© 2016 ООО ВЕБО

E-mail:

info@webo.name

Блог:

blog.webogroup.com

Twitter:

wboptimizer

Продукты

Облачное ускорение сайтов

WEBO Site SpeedUp

WEBO Pulsar

WEBO Enterprise SpeedoMeter

Информация

О компании

Вакансии

Контакты

Карта сайта

Текст ниже — перевод статьи "Unobtrusive Javascript", чей автор Christian Heilmann любезно дал мне согласие на перевод. Оригинальный адрес статьи www.onlinetools.org/articles/unobtrusivejavascript/chapter5.html.

Улучшать формы с помощью JavaScript — это очень хорошая идея. Ничто не расстраивает больше, чем заполнить форму, отправить ее серверу, ожидать пока страница загрузится и получить уведомления, что вы забыли заполнить то или другое поле.

Для пользователя куда приятнее получить ответ немедленно, еще до отправки данных, нам же это уменьшит серверный трафик.

Формы и JavaScript — обманчивая мощь

JavaScript может сделать форму гораздо лучше, увеличив ее юзабилити, но мы должны осознавать несколько вещей:

Нам все же нужно делать проверку на стороне сервера, чтобы неверные данные — посланные пользователем с отключенным JavaScript — не достигали базы данных.

Не существует такой вещи как "автоматически сгенерированный скрипт проверки формы". Каждая форма исключительна и следует определенным правилам валидации и процесса перехода. Учет каждой возможности приведет к чрезмерно раздутому и медленному коду, способ лучше — это повторное использование отдельных частей библиотеки скрипта. Это дает более легкую поддержку и быстрое исполнение.

Попробуйте сохранять способы проверки одинаковыми, как на стороне клиента, так и на стороне сервера. Мы могли бы использовать классы для валидации на стороне клиента, например, <input class="required"/>, но они не отсылаются при отправлении формы.

Код нашей формы

<form action="formsend.php" method="post"

onsubmit="return checkform(this);">

<p>

<input type="hidden" name="required" id="required"

value="name,surname,email,tac,msg,contactform" />

<label for="name">Name</label>

<input type="text" name="name" id="name" /><span>\*</span>

</p>

<p>

<label for="surname">Surname</label>

<input type="text" name="surname" id="surname" /><span>\*</span>

</p>

<p>

<label for="email">Email</label>

<input type="text" name="email" id="email" /><span>\*</span>

</p>

<p>

<label for="phone">Phone number</label>

<input type="text" name="phone" id="phone" />

</p>

<p>

<label for="contactform">Prefered form of contact</label>

<select id="contactform" name="contactform">

<option value="">Please choose</option>

<option value="p">phone</option>

<option value="e">email</option>

</select><span>\*</span>

</p>

<p>

<label for="msg">Your message</label>

<textarea name="msg" id="msg"></textarea><span>\*</span>

</p>

<p>

<input type="checkbox" name="tac" id="tac" />

I have read the <label for="tac">terms and conditions</label>

and agree with them.</label><span>\*</span>

</p>

<p>

<input type="submit" value="Send information" />

</p>

</form>

Это отлично сверстанная форма, выполненная с метками, чтобы приспособить ее для не пользующихся мышью (non-visual) пользователей и тем пользователей, которым может быть трудно попасть в чекбокс их координатно-указательным устройством(мышью, трэкболом и т.д.). Для валидации, у нас есть скрытое поле называемое required (обязательные), оно содержит список всех обязательных для заполнения полей через запятую. Это стандарт для скриптов валидации формы (помните formmail.pl?) с незапамятных времен.

Правила, которым мы хотим следовать:

Убедитесь, что каждое обязательное поле заполнено, выбрано или отмечено.

Убедитесь, что email введен в правильной форме.

Большинство скриптов для валидации выводят список неправильно заполненных обязательных полей в предупреждение (alert) JavaScript. В этом есть смысл, если форма велика и сложна, но предупреждения раздражают и несколько отталкивают. Давайте попробуем для этой небольшой формы другой подход: каждое поле, в котором есть ошибка, должно получить небольшую иконку с предостерегающим знаком и красный цвет фона, а также мы покажем сообщение над кнопкой отправки формы (submit button), в котором говорится, что найдены ошибки.

Наш скрипт проверки формы ( checkform() )

Мы начинаем наш скрипт с тестирования, доступна ли DOM, и существует ли поле с ID required. Если не выполняется хотя бы одно из этих условий, мы возвращаемся к документу, и наш покой будет обеспечивать теперь PHP-скрипт обработки ошибок formsend.php.

function checkform(of)

{

if(!document.getElementById || !document.createTextNode){return;}

if(!document.getElementById('required')){return;}

Продолжаем, определяя все переменные, используемые в отображении ошибки, и разбивая строку — список ID обязательых полей в поле required — в массив.

var errorID='errormsg';

var errorClass='error'

var errorMsg='Please enter or change the fields marked with a ';

var errorImg='img/alert.gif';

var errorAlt='Error';

var errorTitle='This field has an error!';

var reqfields=document.getElementById('required').value.split(',');

Раз мы добавляем элемент с определенным ID в errorID, а также изображения к каждому неправильно заполненному полю, нам нужно удалить все это, когда скрипт будет выполняться во второй раз. Иначе у нас будет несколько сообщений об ошибке и несколько изображений.

// Удаляем старые сообщения

// если есть старое поле errormessage, удаляем его

if(document.getElementById(errorID))

{

var em=document.getElementById(errorID);

em.parentNode.removeChild(em);

}

// удаляем старые изображение и классы

// в обязательных для заполнения полях

for(var i=0;i<reqfields.length;i++)

{

var f=document.getElementById(reqfields[i]);

if(!f){continue;}

if(f.previousSibling && /img/i.test(f.previousSibling.nodeName))

{

f.parentNode.removeChild(f.previousSibling);

}

f.className='';

}

Сейчас мы сделаем то, что потом уберем к прежнему состоянию. Мы перебираем обязательные поля и проверяем, в первую очередь, существует ли поле. Если нет, мы пропускаем один круг цикла. Это делается исключительно, чтобы избежать сообщений об ошибках, в настоящей же форме должны быть все обязательные поля.

// перебираем обязательыне поля

for(var i=0;i<reqfields.length;i++)

{

// проверяем, существует ли это обязательное поле

var f=document.getElementById(reqfields[i]);

if(!f){continue;}

Потом мы проверяем каждое поле в соответствии с его типом. Для текстовых областей и полей мы должны проверить значение, для чекбоксов — отмеченный атрибут, для выпадающих меню — определен ли selectedIndex, и больше ли он 0.

Если в каком-то из полей есть ошибка, мы посылаем ее как объект методу cf\_adderr(). Особый случай — это поле email, так как оно также нуждается в проверке на правильность электронного почтового адреса. Эта проверка исполняется другим методом, называемым cf\_isEmailAddr(), в котором используются регулярные выражения.

// проверяем правильность заполнения обязательных полей

// в соответствии с их типом

switch(f.type.toLowerCase())

{

case 'text':

if(f.value=='' && f.id!='email'){cf\_adderr(f)}

// email — особое поле и требует иной проверки

if(f.id=='email' &&

!cf\_isEmailAddr(f.value)){cf\_adderr(f)}

break;

case 'textarea':

if(f.value==''){cf\_adderr(f)}

break;

case 'checkbox':

if(!f.checked){cf\_adderr(f)}

break;

case 'select-one':

if(!f.selectedIndex && f.selectedIndex==0){cf\_adderr(f)}

break;

}

}

Если какая-либо из проверок выше запускает отчет об ошибке, cf\_adderr() создает сообщение об ошибке (DIV с errorId в качестве ID). Поэтому мы возвращаемся к процессу отправки формы, только когда этого элемента не существует.

return !document.getElementById(errorID);

Это главная функция, на данный момент нам нужно сконцентрироваться на используемых методах, первый это добавление изображений со знаками ошибки и сообщений об ошибках.

Читать дальше

Ненавязчивый JavaScript. Глава 1: Операция «Чистота» в действии

Ненавязчивый JavaScript. Глава 2: Как достичь того, что мы хотим изменить

Ненавязчивый JavaScript. Глава 3: Создание и уничтожение элементов страницы

Ненавязчивый JavaScript. Глава 4: Вызываем ветер (скрипты)