**Делегирование событий**

Всплытие событий позволяет реализовать один из самых важных приёмов разработки – *делегирование*.

Он заключается в том, что если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому – мы ставим один обработчик на их общего предка. Из него можно получить целевой элемент event.target, понять на каком именно потомке произошло событие и обработать его.

**[Пример «Ба Гуа»](http://learn.javascript.ru/event-delegation" \l "primer-ba-gua)**

Рассмотрим пример – [диаграмму «Ба Гуа»](http://en.wikipedia.org/wiki/Ba_gua). Это таблица, отражающая древнюю китайскую философию.

Вот она:

Её HTML (схематично):

<table>

<tr>

<th colspan="3"><em>Bagua</em> Chart: Direction, Element, Color, Meaning</th>

</tr>

<tr>

<td>...<strong>Northwest</strong>...</td>

<td>...</td>

<td>...</td>

</tr>

<tr>...еще 2 строки такого же вида...</tr>

<tr>...еще 2 строки такого же вида...</tr>

</table>

В этой таблице всего 9 ячеек, но могло быть и 99, и даже 9999, не важно.

**Наша задача – реализовать подсветку ячейки <td> при клике.**

Вместо того, чтобы назначать обработчик для каждой ячейки, которых может быть очень много – мы повесим *единый обработчик* на элемент <table>.

Он будет использовать event.target, чтобы получить элемент, на котором произошло событие, и подсветить его.

Код будет таким:

var selectedTd;

table.onclick = function(event) {

var target = event.target; // где был клик?

if (target.tagName != 'TD') return; // не на TD? тогда не интересует

highlight(target); // подсветить TD

};

function highlight(node) {

if (selectedTd) {

selectedTd.classList.remove('highlight');

}

selectedTd = node;

selectedTd.classList.add('highlight');

}

Такому коду нет разницы, сколько ячеек в таблице. Обработчик всё равно один. Я могу добавлять, удалять <td> из таблицы, менять их количество – моя подсветка будет стабильно работать, так как обработчик стоит на <table>.

Однако, у текущей версии кода есть недостаток.

**Клик может быть не на том теге, который нас интересует, а внутри него.**

В нашем случае, если взглянуть на HTML таблицы внимательно, видно, что ячейка содержит вложенные теги, например <strong>:

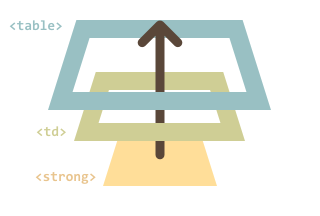
<td>

<strong>Northwest</strong>

...Metal..Silver..Elders...

</td>

Естественно, клик может произойти внутри <td>, на элементе <strong>. Такой клик будет пойман единым обработчиком, но target у него будет не <td>, а <strong>:



Внутри обработчика table.onclick мы должны по event.target разобраться, в каком именно <td> был клик.

Для этого мы, используя ссылку parentNode, будем идти вверх по иерархии родителей от event.target и выше и проверять:

* Если нашли <td>, значит это то что нужно.
* Если дошли до элемента table и при этом <td> не найден, то наверное клик был вне <td>, например на элементе заголовка таблицы.

Улучшенный обработчик table.onclick с циклом while, который это делает:

table.onclick = function(event) {

var target = event.target;

// цикл двигается вверх от target к родителям до table

while (target != table) {

if (target.tagName == 'TD') {

// нашли элемент, который нас интересует!

highlight(target);

return;

}

target = target.parentNode;

}

// возможна ситуация, когда клик был вне <td>

// если цикл дошёл до table и ничего не нашёл,

// то обработчик просто заканчивает работу

}

**На заметку:**

Кстати, в проверке while можно бы было использовать this вместо table:

while (target != this) {

// ...

}

Это тоже будет работать, так как в обработчике table.onclick значением this является текущий элемент, то есть table.

Можно для этого использовать и метод closest, при поддержке браузером:

table.onclick = function(event) {

var target = event.target;

var td = target.closest('td');

if (!td) return; // клик вне <td>, не интересует

// если клик на td, но вне этой таблицы (возможно при вложенных таблицах)

// то не интересует

if (!table.contains(td)) return;

// нашли элемент, который нас интересует!

highlight(td);

}

**[Применение делегирования: действия в разметке](http://learn.javascript.ru/event-delegation" \l "primenenie-delegirovaniya-deystviya-v-razmetke)**

Обычно делегирование – это средство оптимизации интерфейса. Мы используем один обработчик для *схожих*действий на однотипных элементах.

Выше мы это делали для обработки кликов на <td>.

**Но делегирование позволяет использовать обработчик и для абсолютно разных действий.**

Например, нам нужно сделать меню с разными кнопками: «Сохранить», «Загрузить», «Поиск» и т.д. И есть объект с соответствующими методами: save, load, search и т.п…

Первое, что может прийти в голову – это найти каждую кнопку и назначить ей свой обработчик среди методов объекта.

Но более изящно решить задачу можно путем добавления одного обработчика на всё меню, а для каждой кнопки в специальном атрибуте, который мы назовем data-action (можно придумать любое название, но data-\* является валидным в HTML5), укажем, что она должна вызывать:

<button data-action="save">Нажмите, чтобы Сохранить</button>

Обработчик считывает содержимое атрибута и выполняет метод. Взгляните на рабочий пример:

<div id="menu">

<button data-action="save">Сохранить</button>

<button data-action="load">Загрузить</button>

<button data-action="search">Поиск</button>

</div>

<script>

function Menu(elem) {

this.save = function() {

alert( 'сохраняю' );

};

this.load = function() {

alert( 'загружаю' );

};

this.search = function() {

alert( 'ищу' );

};

var self = this;

elem.onclick = function(e) {

var target = e.target;

var action = target.getAttribute('data-action');

if (action) {

self[action]();

}

};

}

new Menu(menu);

</script>

Обратите внимание, как используется трюк с var self = this, чтобы сохранить ссылку на объект Menu. Иначе обработчик просто бы не смог вызвать методы Menu, потому что его собственный this ссылается на элемент.

Что в этом случае нам дает использование делегирования событий?

* Не нужно писать код, чтобы присвоить обработчик каждой кнопке. Меньше кода, меньше времени, потраченного на инициализацию.
* Структура HTML становится по-настоящему гибкой. Мы можем добавлять/удалять кнопки в любое время.
* Данный подход является семантичным. Также можно использовать классы .action-save, .action-load вместо атрибута data-action.

**[Итого](http://learn.javascript.ru/event-delegation" \l "itogo)**

Делегирование событий – это здорово! Пожалуй, это один из самых полезных приёмов для работы с DOM. Он отлично подходит, если есть много элементов, обработка которых очень схожа.

Алгоритм:

1. Вешаем обработчик на контейнер.
2. В обработчике: получаем event.target.
3. В обработчике: если event.target или один из его родителей в контейнере (this) – интересующий нас элемент – обработать его.

Зачем использовать:

* Упрощает инициализацию и экономит память: не нужно вешать много обработчиков.
* Меньше кода: при добавлении и удалении элементов не нужно ставить или снимать обработчики.
* Удобство изменений: можно массово добавлять или удалять элементы путём изменения innerHTML.

Конечно, у делегирования событий есть свои ограничения.

* Во-первых, событие должно всплывать. Нельзя, чтобы какой-то промежуточный обработчик вызвал event.stopPropagation() до того, как событие доплывёт до нужного элемента.
* Во-вторых, делегирование создает дополнительную нагрузку на браузер, ведь обработчик запускается, когда событие происходит в любом месте контейнера, не обязательно на элементах, которые нам интересны. Но обычно эта нагрузка настолько пустяковая, что её даже не стоит принимать во внимание.