jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX.

Возможности

- Движок кроссбраузерных CSS-селекторов Sizzle[1], выделившийся в отдельный проект;
- Переход по дереву DOM, включая поддержку XPath как плагина;
- События;
- Визуальные эффекты;
- AJAX-дополнения;
- JavaScript-плагины.

Точно так же, как CSS отделяет визуализацию от структуры HTML, JQuery отделяет поведение от структуры HTML. Например, вместо прямого указания на обработчик события нажатия кнопки управление передаётся JQuery, которая идентифицирует кнопки и затем преобразует его в обработчик события клика. Такое разделение поведения и структуры также называется принципом ненавязчивого JavaScript.

Библиотека jQuery содержит функциональность, полезную для максимально широкого круга задач. Тем не менее, разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всюду, поскольку это привело бы к большому коду, бóльшая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов[2]. Это позволяет собрать для ресурса именно ту JavaScript-функциональность, которая на нём была бы востребована.

Простейший пример

```
<script>
    // ждём загрузки всего документа
    // после этого будет выполнена анонимная функция
    // которую мы передали в качестве параметра
    jQuery(document).ready(function(){
```

```
        jQuery("h2").css("color", "red");
    });
</script>
```

Также можно использовать сокращённый вариант без явного вызова метода ready():

```
<script>
    $(function(){
        $("h2").css("color", "red");
    });
</script>
```

Селекторы

Как говорилось ранее, в поиске элементов на странице заключается практически половина успешной работы с jQuery.

выбор элементов по «id» либо «className» аналогично используемым в CSS:

```
$("#content") // выбираем элемент с id=content
$("div#content") // выбираем div с id=content (хотя id и так однозначен)
$(".wrapper") // выбираем элементы с class=wrapper
$("div.wrapper") // выбираем div'ы с class=wrapper
$(".wrapper.box") // выбираем элементы с class=wrapper и box
$("h2") // выбираем все теги h2
$("h1, h2") // выбираем все теги h1 и h2
```

Теперь вспомним, что мы в DOMe не одни, это таки иерархическая структура:

```
$("article h2") // выбираем все теги h2 внутри тега article
 $("div article h2") // выбираем все теги h2 внутри тега article // внутри тега div
$("article").find("h2") // аналогично примерам выше
```

$("div").find("article").find("h2") //

У нас есть соседи:

$("h1 + h2") // выбор всех h2 элементов, перед которыми идут h1 // элементы (у нас только один такой)

$("#stick ~ article") // выбор всех article элементов после элемента // с id=stick

$("#stick").prev() // выбор предыдущего элемента от найденного

$("#stick").next() // выбор следующего элемента от найденного

Родственные связи:

$("*") // выбор всех элементов

$("article > h2") // выбираем все теги h2 которые являются // непосредственными потомками тега article

$("article > *") // выбор всех потомков элементов p

$("article").children()// --

$("p").parent() // выбор всех прямых предков элементов p

$("p").parents() // выбор всех предков элементов p (не понадобится)

$("p").parents("div") // выбор всех предков элемента p которые есть div // parents принимает в качестве параметра селектор

Поиск по атрибутам

Ещё со времён CSS2 была возможность найти элемент с определёнными атрибутами, в CSS3 добавили ещё возможностей по поиску:

a[href] — все ссылки с атрибутом href

a[href=#] — все ссылки с href=#

a[href~=#] — все ссылки с # где-то в href

a[hreflang|=en] — все ссылки, для которых hreflang начинается с en и обрезается по символу «-» — en, en-US, en-UK

a[href^=http] — ссылки начинающиеся с http

a[href*=google.com] — ссылки на погуглить

a[href$=.pdf] — ссылки на PDF файлы (по идее)

Заглянув внутрь jQuery вы скорей всего найдёте то место, где ваше выражение будет анализироваться с помощью регулярных выражений, по этой причине в селекторах необходимо экранировать специальные символы используя обратный слеш «\\»: $("a[href^=\\/]").addClass("internal");

Поиск по дочерним элементам

Хотелось бы еще обратить внимание на продвинутые селекторы из спецификации CSS3 — много интересных:

:first-child – первый дочерний элемент

:last-child – последний дочерний элемент

:nth-child(2n+1) – выборка элементов по несложному уравнению

:not(…) – выбрать те, что не подпадают под вложенную выборку

Но поскольку не все браузеры знакомы с CSS3-селекторами, то мы можем использовать jQuery для назначения стилей:

$("div:last-child").addClass("last-paragraph");

Атрибуты элементов и CSS

В предыдущих примерах мы уже изменяли CSS-свойства DOM-элементов, используя одноименный метод css(), но это далеко не всё. Теперь копнём поглубже, чтобы не штурмовать форумы банальными вопросами. Копать начнём с более досконального изучения метода css(): css(property) — получение значения CSS свойства

css(property, value) — установка значения CSS свойства

css({key: value, key:value}) — установка нескольких значений

css(property, function(index, value) { return value }) — тут для установки значения используется функция обратного вызова, index это порядковый номер элемента в выборке, value — старое значение свойства (в просторечии — callback-функция)

Метод css() возвращает текущее значение, а не прописанное в CSS файле по указанному селектору Примеры использования (css.html):

$("#my").css('color') // получаем значение цвета шрифта

$("#my").css('color', 'red') // устанавливаем значение цвета шрифта

// установка нескольких значений

$("#my").css({ 'color':'red', 'font-size':'14px', 'margin-left':'10px' })

// альтернативный способ

$("#my").css({ color:'red', fontSize:'14px', marginLeft:'10px', })

// используя функцию обратного вызова

$("#my").css('height', function(i, value){ return parseFloat(value) * 1.2; })

Стоит ещё описать манипуляции с классами, тоже из разряда первичных навыков:

addClass(className) — добавление класса элементу

addClass(function(index, currentClass){ return className }) — добавление класса используя функцию обратного вызова

hasClass(className) — проверка на причастность к определённому классу

removeClass(className) — удаление класса

removeClass(function(index, currentClass){ return className }) — удаление класса используя функцию обратного вызова

toggleClass(className) — переключение класса toggleClass(className, switch) — переключение класса по флагу switch t

В приведённых функциях в качестве className может быть строка содержащая список классов через пробел.

Также, стоит вспомнить, что у DOM элементов бывают атрибуты отличные от класса, и мы их тоже можем изменять, для этого нам потребуются следующие методы:

attr(attrName) — получение значения атрибута

attr(attrName, attrValue) — установка значения атрибута (также можно использовать hash, либо функцию обратного вызова)

removeAttr(attrName) — удаление атрибута

Атрибуты, с которыми вам чаще других придётся сталкиваться:

// получение альтернативного текста картинки var altText = $('img').attr('alt')

// изменение адреса картинки $('img').attr('src', '/images/default.png')

// работаем со ссылками $('a#my').attr({ 'href':'http://anton.shevchuk.name', 'title':'My Personal Blog', });

Кроме атрибутов, также есть свойства элементов, к ним относится selectedIndex, tagName, nodeName, nodeType, ownerDocument, defaultChecked и defaultSelected. Список невелик, можно и запомнить. Для работы со свойствами используем функции из семейства prop():

prop(propName) — получение значения свойства

prop(propName, propValue) — установка значения свойства (также можно использовать hash, либо функцию обратного вызова)

removeProp(propName) — удаление свойства (скорей всего никогда не понадобится)

Стоит определиться, что же из себя представляют события web-страницы. Так вот – события – это любые действия пользователя, будь то ввод данных с клавиатуры, проматывание страницы или передвижения мышки, и конечно же «клики».

jQuery работает практически со всеми событиями в JavaScript'e, приведу список оных с небольшими пояснениями:

change — изменение значения элемента (значение, при потери фокуса, элемента отличается от изначального, при получении фокуса)

click — клик по элементу (порядок событий: mousedown, mouseup, click)

dblclick — двойной щелчок мышки

resize — изменение размеров элементов

scroll — скроллинг элемента

select — выбор текста (актуален только для input[type=text] и textarea)

submit — отправка формы

focus — фокус на элементе - актуально для input[type=text], но в современных браузерах работает и с другими элементами

blur — фокус ушёл с элемента — актуально для input[type=text] — срабатывает при клике по другому элементу на странице или по событию клавиатуры (к примеру переключение по tab'y)

Вызов большинства из перечисленных событий можно эмулировать непосредственно из
самого скрипта:

```
<script>
	$("#menu li a").click()
	// или используя метод trigger
	$("#menu li a").trigger("click")
</script>
```

Теперь стоит рассказать немного и об обработчиках событий, для примера возьму код
строкой выше, и слегка его модифицирую:

```
$("#menu li a").click(function(event){
	alert("Hello!")
})
```

Теперь кликнув по ссылке вы увидите приветствие и после закрытия оного браузер
перейдет по ссылке указанной в атрибуте href. Но это не совсем то, что мне хотелось –
надо было лишь вывести текст, и никуда не уходить. Ага, для этого стоит отменить действие
по умолчанию:

```
$("#menu li a").click(function(event){
	alert("Hello!");
	event.preventDefault();
})
```

Теперь перехода нет, т.к. метод preventDefault() предотвращает данное действие. Но вот
если кто-то повесит обработчик на само меню?

```
$("#menu").click(function(event){
	alert("Menu!");
})
```

В результате мы получим два сообщения, но почему? Если у вас возникает подобный вопрос, значит вы еще не знакомы с тем, как обрабатываются события. Попробую кратенько дать вводную, когда вы кликаете на элементе в DOM дереве, то происходит «погружение» события – т.е. вначале все родительские элементы могут обработать «клик», и лишь потом он доберётся до элемента по которому был совершён, но и это еще не всё, затем событие начинает проделывать обратный путь – «всплывает», давая тем самым
второй шанс родительским элементам обработать событие.

Хорошо, вроде бы понятно, теперь вернёмся к нашему примеру, и пытаемся понять что же у нас происходит – у нас есть обработчик клика для ссылки и непосредственно для самого меню, в котором эта ссылка находится. Теперь кликая по ссылке, срабатывает обработчик события на ссылке, и затем событие всплывает до меню, и срабатывает его обработчик события click. Но это не совсем желаемый результат, и для борьбы с подобным вредительством, необходимо останавливать «всплытие» событий:

```
$("#menu li a").click(function(event){
	alert("Hello!");
	event.preventDefault(); event.stopPropagation();
})
```

Для ускорения разработки в jQuery есть быстрый способ вызова этих двух методов за раз:

```
$("#menu li a").click(function(event){
alert("Hello!");
return false; // вот это он :)
})
```

Теперь у вас есть достаточный багаж знаний, чтобы легко манипулировать событиями на странице. Хотя я добавлю еще немного — для того, чтобы

сработал лишь ваш обработчик события, можно использовать метод stopImmediatePropagation():

```
 $("#menu li a").click(function(event){
        alert("Hello!");
        event.stopImmediatePropagation();
        return false;
 })
$("#menu li a").click(function(event){
        alert("Hello again!");
        return false;
 })
```

В данном примере, при клике по ссылке будет выведено лишь одно сообщение. И да, порядок имеет значение.

Практическое задание

Взяв за основы предыдущие работы полностью перепишите манипулирование элементами, изменение свойств и атрибутов и привязку событий с помощью jQuery