Современный учебник JavaScript

© Илья Кантор

Сборка от 27 апреля 2014 для печати

Внимание, эта сборка может быть устаревшей и не соответствовать текущему тексту.
Актуальный онлайн-учебник, с интерактивными примерами, доступен по адресу http://learn.javascript.ru.

Вопросы по JavaScript можно задавать в комментариях на сайте или на форуме javascript.ru/forum.

Вопросы по сборке, предложения по её улучшению – можно писать мне, по адресу iliakan@javascript.ru .

Глава: Разное

В файле находится только одна глава учебника. Это сделано в целях уменьшения размера файла, для удобного чтения с устройств.

Содержание

Мини-библиотека функций учебника

Здесь описаны важные вспомогательные функции, применяемые в примерах учебника.

Их количество сведено к минимуму, чтобы примеры были более понятными.

Функции DOM: `dom.js`

Минимально необходимые функции для работы с DOM находятся в файле dom.js [1]:

addClass/removeClass/hasClass(elem, cls) [2]
: Добавляют/удаляют/проверяют CSS-класс элемента

fixEvent(e, _this) [3]
: Нужна только для IE<9. Исправляет объект события, добавляя стандартные свойства. Второй аргумент, если он передан, записывается в `e.currentTarget`.

  Как правило, эта функция находится в начале обработчика события.

getChar(event) [4]
: Кросс-браузерно возвращает символ для события `keypress` или `null`, если была нажата специальная клавиша.

getCoords(elem) [5]
: Возвращает объект с координатами элемента относительно документа `left/top`.

При решении задач для поиска элементов допустимо использование querySelector [6] и querySelectorAll [7]. Таким образом, ваш код будет работать во всех современных браузерах, исключая IE<8. Это для простоты, подразумевается, что вы сможете написать и код без этих методов. Тем более, что учебник также содержит описание других способов поиска в DOM и задачи на них.

Функции ООП: `misc.js`

Библиотека для функций и ООП: misc.js [8]:

bind(func, context) [9]
: Возвращает обёртку, которая вызывает функцию `func` в контексте `this = context`.

  Можно также добавить аргументы: `bind(func, context, arg1, arg2...)`, более подробно см. описание [10].

copy(dst, src1, src2...) [11]
: Копирует все свойства объектов `src*` в объект `dst`, с перезаписью.

inherit(proto) [12]
: Создаёт пустой объект с прототипом `proto`.

Шаблонка: `tmpl.js`

Файл tmpl.js [13] содержит функции:

`esc(text)`
: Заменяет спецсимволы HTML в строке `text`.

tmpl(str) [14]
: Компилирует шаблон из строки `str`.

Более подробно — читайте в статье Шаблонизация в JavaScript [15].

Анимация `animate.js`

Функция animate [16] находится в файле animate.js [17]. Механизм её работы описан в главе JS-Анимация [18].

В файле находятся также `delta`-функции и преобразования `makeEaseOut/makeEaseInOut`.

Выделение: Range, TextRange и Selection

В этой статье речь пойдет о документированных, но нечасто используемых объектах `Range`, `TextRange` и `Selection`. Мы рассмотрим вольный перевод спецификаций с понятными примерами и различные кроссбраузерные реализации.

Range

`Range` — это объект, соответствующий фрагменту документа, который может включать узлы и участки текста из этого документа. Наиболее подробно объект `Range` описан в спецификации DOM Range [19] .

Чтобы понять о чем речь, обратимся к самому простому случаю `Range`, который будет подробно рассмотрен ниже — к выделениям. В приводимом ниже примере выделите несколько слов в предложении. Будет выводиться текстовое содержимое выделяемой области:

> Соберем микс из **жирности**, *курсива* и ссылки [20] и повыделяем здесь.

Но такие области можно создавать не только с помощью пользовательского выделения, но и из JavaScript-сценария, выполняя с ними определенные манипуляции. Однако, написать простой иллюстрирующий код сразу не выйдет, т.к. есть одно НО — Internet Explorer до версии 9. В Microsoft создали собственную реализацию — объект TextRange [21] . Разбёрем каждую реализацию по-отдельности.

DOM-реализация Range (кроме IE<9)

`Range` состоит из двух граничных точек (boundary-points), соответствующих началу и концу области. Позиция любой граничной точки определяется в документе с помощью двух свойств: узел (node) и смещение (offset).

Контейнером (container) называют узел, содержащий граничную точку. Сам контейнер и все его предки называются родительскими контейнерами (ancestor containers) для граничной точки. Родительский контейнер, включающий обе граничные точки, называют корневым

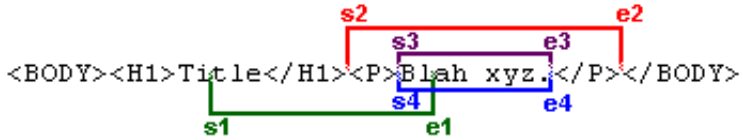
контейнером (root container).

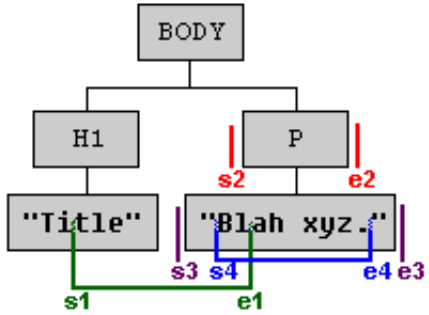
```
<body><h1>Title</h1><p>Blah xyz</p></body>
```

На изображении выше граничные точки выделения лежат в текстовых узлах (`#text1` и `#text2`), которые являются контейнерами. Для левой границы родительскими контейнерами являются `#text1`, `H1`, `BODY`, для правой — `#text2`, `P`, `BODY`. Общий родитель для обоих граничных точек — `BODY`, этот элемент является корневым контейнером.

Если контейнер является текстовым узлом, то смещение определяется в символах от начала DOM-узла. Если контейнер является элементом (`Document`, `DocumentFragment`, `Element`...), то смещение определяется в дочерних узлах.

Смотрим на иллюстрацию (источник [22]):





Range	s		e	
	Node	Offset	Node	Offset
s1—e1	Text1	2	Text2	2
s2—e2	BODY	1	BODY	2
s3—e3	P	0	P	1
s4—e4	Text2	0	Text2	9

Граничные точки объекта `Range` **s1** лежат в текстовых узлах, поэтому смещение задается в символах от начала узла. Для **s2** граничные точки расставлены так, что включают весь абзац Blah xyz , поэтому контейнером является элемент `BODY`, и смещение считается в позициях дочерних узлов.

Объекты `Range` создаются с помощью вызова `document.createRange()`. Объект при этом создается пустой, и граничные точки нужно задать далее его методами `setStart` и `setEnd`. Смотрим пример.

HTML:

```html
<div id="ex2">
  <h2>Соз|даем объект `Range`</h2>
  <p>От третье|го символа заголовка до десятого символа это абзаца.</p>
</div>

<button onclick="alert(domRangeCreate())">
  Создать Range и вывести его текст
</button>

<script>
function domRangeCreate() {
  // Найдем корневой контейнер
  var root = document.getElementById('ex2');
  // Найдем контейнеры граничных точек (в данном случае тестовые)
  var start = root.getElementsByTagName('h2')[0].firstChild;
  var end = root.getElementsByTagName('p')[0].firstChild;
  if (root.createRange) {
    // Создаем Range
    var rng = root.createRange();
    // Задаем верхнюю граничную точку, передав контейнер и смещение
    rng.setStart( start, 3 );
    // Аналогично для нижней границы
    rng.setEnd( end, 10 );
    // Теперь мы можем вернуть текст, который содержится в полученной области
    return rng.toString();
  } else {
    return 'Вероятно, у вас IE<9, смотрите реализацию TextRange ниже';
  }
}
</script>
```

В действии:

Соз|даем Range-объект

От третье|го символа заголовка до десятого символа это абзаца.

Создать Range и вывести его текст

Рассмотрим вкратце свойства и методы Range [23] :

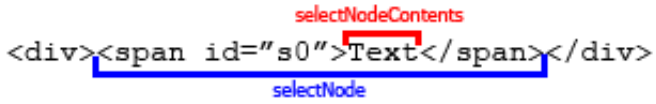
- Свойство `commonAncestorContainer` вернет ссылку на наиболее вложенный корневой контейнер.
- Свойство `startContainer` (`endContainer`) вернет ссылку на контейнер верхней (нижней) граничной точки.
- Свойство `startOffset` (`endOffset`) вернет смещение для верхней (нижней) граничной точки.
- Свойство `collapsed` вернет `true`, если граничные точки имеют одинаковые контейнеры и смещение (`false` в противном случае).

- Метод `setStart` (`setEnd`) задает контейнер (ссылка на узел) и смещение (целочисленное значение) для соответствующих граничных точек. Пример выше.
- Методы `setStartBefore`, `setStartAfter`, `setEndBefore`, `setEndAfter` принимают в качестве единственного аргумента ссылку на узел и устанавливают граничные точки в соот-ии с естественной границей переданного узла. Например:

```
<span id="s1">First</span>
<span id="s2">Second</span>
```

```
var rng = document.createRange();
// Установит верхнюю граничную точку по левой границе спана #s1
rng.setStartBefore( document.getElementById('s1') );
// Установит нижнюю граничную точку по правой границе спана #s2
rng.setEndAfter( document.getElementById('s2') );
```

- Методы `selectNode` и `selectNodeContents` позволяют создать объект `Range` по границам узла, ссылку на который они принимают в качестве единственного аргумента. При использовании `selectNode` передаваемый узел также войдет в `Range`, в то время как `selectNodeContents` создаст объект только из содержимого узла:



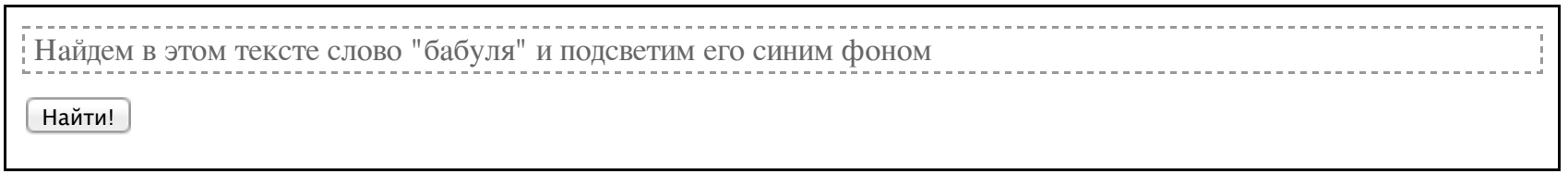
- Метод `collapse` объединяет граничные точки объекта `Range`. В качестве единственного аргумента принимает булево значение (`true` — для объединения в верхней точке, `false` — в нижней). По-умолчанию `true`.
- Метод `toString` вернет текстовое содержимое объекта `Range`.
- Метод `cloneContents` вернет копию содержимого объекта `Range` в виде фрагмента документа.
- Метод `cloneRange` вернет копию самого объекта `Range`.
- Метод `deleteContents` удаляет всё содержимое объекта `Range`.
- Метод `detach` извлекает текущий объект из DOM, так что на него больше нельзя сослаться.
- Метод `insertNode` принимает в качестве единственного аргумента ссылку на узел (или фрагмент документа) и вставляет его в содержимое объекта `Range` в начальной точке.
- Метод `extractContents` вырезает содержимое объекта `Range` и возвращает ссылку на полученный фрагмент документа.
- Метод `surroundContents` помещает всё содержимое текущего объекта `Range` в новый родительский элемент, ссылка на который принимается в качестве единственного аргумента.
- Метод `compareBoundaryPoints` используется для сравнения граничных точек.

Для примера решим небольшую задачку. Найдём в текстовом узле фразу и подсветим её синим фоном.

```html
<div id="ex3">
  Найдем в этом тексте слово "бабуля" и подсветим его синим фоном
</div>

<script>
function domRangeHighlight(text) {
  // Получим текстовый узел
  var root = document.getElementById('ex3').firstChild;
  // и его содержимое
  var content = root.nodeValue;
  // Проверим есть ли совпадения с переданным текстом
  if ( ~content.indexOf( text ) ) {
    if ( document.createRange ) {
      // Если есть совпадение, и браузер поддерживает Range, создаем объект
      var rng = document.createRange();
      // Ставим верхнюю границу по индексу совпадения,
      rng.setStart( root, content.indexOf( text ) );
      // а нижнюю по индексу + длина текста
      rng.setEnd( root, content.indexOf( text ) + text.length );
      // Создаем спан с синим фоном
      var highlightDiv = document.createElement('span');
      highlightDiv.style.backgroundColor = 'blue';
      // Обернем наш Range в спан
      rng.surroundContents( highlightDiv );
    } else {
      alert('Вероятно, у вас IE<9, смотрите реализацию TextRange ниже');
    }
  } else {
    alert('Совпадений не найдено');
  }
}
</script>
```

В действии:



С остальными свойствами и методами поэкспериментируйте сами. Перейдем к реализации range в IE.

`TextRange` (для IE)

Объект `TextRange` в реализации MSIE — это текстовый диапазон нулевой и более длины. У данного диапазона также есть свои границы, «перемещать» которые можно на целое число текстовых единиц: character(символ), word (слово), sentence (предложение). То есть можно взять и сдвинуть границу на 2(5, 8 и т.д.) слова (символа, предложения) вправо (влево). При этом у объекта сохраняются данные о HTML-содержимом диапазона и есть методы взаимодействия с DOM.

Объект `TextRange` создается с помощью метода `createTextRange`, который можно вызывать в контексте элементов `BODY`, `BUTTON`, `INPUT` (большинство типов), `TEXTAREA`.

Простой пример с кнопкой:

```
<input id="buttonId" type="button" value="Test button" onclick="alert( ieTextRangeCreate() );" />

<script>
function ieTextRangeCreate() {
  // Найдем кнопку
  var button = document.getElementById('buttonId');
  // Если мы в ИЕ
  if ( button.createTextRange && button.createTextRange() != undefined ) {
    // Создаем TextRange
    var rng = button.createTextRange();
    // И вернем текстовое содержимое полученного объекта
    return rng.text;
  } else {
    return 'Вероятно, у вас не IE, смотрите реализацию Range выше';
  }
}
</script>
```

Рассмотрим свойства и методы объекта TextRange [24] (не все, только самые необходимые):

- Свойство `boundingWidth` (boundingHeight) вернет ширину (высоту), которую занимает объект TextRange в пикселях.
- Свойство `boundingTop` (`boundingLeft`) вернет Y(X)-координату верхнего левого угла тестовой области относительно окна документа.
- Свойство `htmlText` вернет HTML-содержимое объекта.
- Свойство `text` вернет текстовое содержимое объекта (см. пример выше).
- Свойство `offsetTop` (`offsetLeft`) вернет Y(X)-координату верхнего левого угла тестовой области относительно предка.

- Метод `collapse` объединяет граничные точки диапазона. В качестве единственного аргумента принимает булево значение (`true` — для объединения в верхней точке, `false` — в нижней). По-умолчанию true.
- Метод `duplicate` клонирует имеющийся текстовый диапазон, возвращая новый, точно такой же.
- Метод `expand` расширяет текущий тектовый диапазон до единицы текста, переданной в качестве единственного текстового аргумента:
  - `"character'` — символ.
  - `"word"` — слово
  - `"sentence"` — предложение
  - `"textedit"` — сворачивает до первоначального диапазона.

  Вернет `true` (`false`) в случае успеха (неудачи).

- Метод `findText` ищет в диапазоне совпадения с текстовой строкой, передаваемой в качестве первого аргумента (без учета регистра). Если совпадение найдено, то границы диапазона сворачиваются до него. В качестве второго (необязательного) аргумента можно передать целое число, указывающее число символов от верхней точки, в которых нужно производить поиск. Далее в качестве аргументов можно перечислять INT-флаги, которые вам вряд ли понадобятся [25] .
- Метод `getBookmark` возвращает в случае успешного вызова строку, по которой можно будет восстановить текущее состояние текстового диапазона с помощью метода `moveToBookmark`.
- Метод `inRange` принимает в качестве аргумента другой `TextRange` и проверяет, входит ли его текстовый диапазон в диапазон контекстного объекта. Возвращает булево значение.

- Метод `isEqual` проверяет является ли текущий `TextRange` идентичным переданному в качестве аргумента. Возвращает булево значение.
- Метод `move(sUnit [, iCount])` сворачивает текущий диапазон до нулевой длины и передвигает на единицу текста, переданного в качестве первого аргумента (character | word | sentence | textedit). В качестве второго (необязательного) аргумента можно передать число единиц, на которое следует передвинуть диапазон.
- Метод `moveEnd` (`moveStart`), аналогично методу move, передвигает верхнюю (нижнюю) границу диапазона на единицу текста, число которых также можно задать необязательным вторым параметром.
- Метод `moveToElementText` принимает в качестве аргумента ссылку на DOM-элемент и выставляет границы диапазона Textобъекта `Range` по границам полученного элемента.
- Метод `moveToPoint` принимает в качестве двух обязательных аргументов X и Y-координаты (в пикселях) относительно верхнего левого угла документа и переносит границы диапазона туда.
- Метод `parentElement` вернет ссылку на элемент, который полностью содержит диапазон объекта `TextRange` (или `null`).
- Метод `pasteHTML` заменяет HTML-содержимое текущего текстового диапазона на строку, переданную в качестве единственного аргумента.
- Метод `select` формирует выделение на основе содержимого объекта `TextRange`, о чем мы подробнее поговорим ниже.
- Метод `setEndPoint` принимает в качестве обязательных аргументов текстовый указатель и ссылку на другой `TextRange`, устанавливая в зависимости от значения указателя границы диапазона. Указатели могут быть следующими: ‘StartToEnd’, ‘StartToStart’, ‘EndToStart’, ‘EndToEnd’.

Также к `TextRange` применимы команды метода execCommand [26] , который умеет делать текст жирным, курсивным, копировать его в буфер обмена (только IE) и т.п.

Для закрепления сделаем задачку по поиску текстового содержимого, аналогичную той, что была выше:

```
<div id="ex4">
  Найдем в этом тексте слово "бабуля" и подсветим его синим фоном
</div>

<script>
function ieTextRangeHighlight(text) {
  // Получим ссылку на элемент, в котором будет происходить поиск
  var root = document.getElementById('ex4');
  // Получим значение его текстового потомка
  var content = root.firstChild.nodeValue;
  // Если есть совпадение
  if ( ~content.indexOf(text) ) {
    // и мы в MSIE
    if ( document.body.createTextRange ) {
      // Создадим объект TextRange
      var rng = document.body.createTextRange();
      // Свернем его до root
      rng.moveToElementText( root );
      // Найдем текст и свернем диапазон до него
      if ( rng.findText( text ) )
        // Заменим текстовый фрагмент на span с синим фоном
        rng.pasteHTML( '<span style="background:blue;">' + text + '</span>' );
    } else {
      alert('Вероятно, у вас не  IE, смотрите реализацию Range выше');
    }
  } else {
    alert('Совпадений не найдено');
  }
}
</script>
```

В действии:

Найдем в этом тексте слово "бабуля" и подсветим его синим фоном

Найти!

С остальными свойствами и методами поэкспериментируйте сами.

Selection

Всем знакомо выделение элементов на странице, когда, зажав левую кнопку мыши и передвигая курсор, мы выделяем нужный фрагмент. Или зажимаем Shift и жмём на стрелочки клавиатуры. Или еще как-то, неважно. В данной части статьи мы кроссбраузерно научимся решать две задачи: получать пользовательское выделение и устанавливать собственное.

Получаем пользовательское выделение

Эту задачу мы уже решали в самом начале статьи в примере с миксом [27]. Теперь рассмотрим код:

```
function getSelectionText() {
  var txt = '';
  if (txt = window.getSelection) // Не IE, используем метод getSelection
    txt = window.getSelection().toString();
  } else { // IE, используем объект selection
    txt = document.selection.createRange().text;
  }
  return txt;
}
```

Все браузеры, кроме IE<9 поддерживают метод `window.getSelection()`, который возвращает объект, схожий с рассмотренным ранее `Range`. У этого объекта есть точка начала выделения (anchor) и фокусная точка окончания (focus). Точки могут совпадать. Рассмотрим свойства и методы объекта `Selection`:

- Свойство `anchorNode` вернет контейнер, в котором начинается выделение. Замечу, что началом выделения считается та граница, от которой вы начали выделение. То есть, если вы выделяете справа налево, то началом будет именно правая граница. Это правило работает везде, кроме браузера Opera, в котором `anchorNode` вернет ссылку на узел левого края выделения.
- Свойство `anchorOffset` вернет смещение для начала выделения в пределах контейнера `anchorNode`.
- Свойства `focusNode` и `focusOffset` работают аналогично для фокусных точек, то есть точек окончания выделения. Opera и здесь отличилась, возвращает вместо фокусной точки узел правого края выделения.
- Свойство `rangeCount` возвращает число объектов `Range`, которые входят в полученное выделение. Это свойство полезно при использовании метода `addRange`.

- Метод `getRangeAt` принимает в качестве аргумента индекс объекта `Range` и возвращает сам объект. Если `rangeCount == 1`, то работать будет только `getRangeAt(0)`. Таким образом, мы можем получить объект `Range`, полностью соответствующий текущему выделению.
- Метод `collapse` сворачивает выделение в точку (каретку). Методу можно передать в качестве первого аргумента узел, в который нужно поместить каретку.
- Метод `extend` принимает в качестве аргументов ссылку на контейнер и смещение (`parentNode`, `offset`), и перемещает фокусную точку в это положение.
- Метод `collapseToStart` (`collapseToEnd`) перемещает фокусную (начальную) границу к начальной (фокусной), тем самым сворачивая выделение в каретку.
- Метод `selectAllChildren` принимает в качестве единственного аргумента ссылку на узел и добавляет всех его потомков в выделение.
- Метод `addRange` принимает в качестве аргумента объект `Range` и добавляет его в выделение. Таким образом можно увеличить количество объектов `Range`, число которых нам подскажет свойство `rangeCount`.
- Метод `removeRange` (`removeAllRanges`) удаляет переданный (все) объект `Range` из выделения.
- Метод `toString` вернет текстовое содержимое выделения.

IE предоставляет собственный интерфейс взаимодействия с выделениями — объект `selection` в контексте document. Для работы с этим объектом используются следующие методы:

- Метод `clear` убирает выделение вместе с содержимым.
- Метод `createRange` (ВАЖНО! Не путать со стандартным методом `document.createRange()` для создания объектов `Range`!) создает из содержимого выделения `TextRange`.
- Метод `empty` убирает выделение, но оставляет содержимое.

Надеюсь, теперь, после знакомства с обеими реализациями выделений, код выше стал более понятен.

Установка собственного выделения

Допустим, вам хочется, чтобы какой-то текстовый фрагмент на странице был выделен, как пользовательское выделение. Это нужно при клиентской реализации поиска по странице и некоторых других задач.

Проще всего решить эту задачу следующим образом:

1. Создать объект `Range` (`TextRange` для IE<9).
2. Перевести полученный объект в выделение.

Смотрим реализацию:

```
<div id="ex5">
  Снова будем выделять <span>бабулю</span>, на этот раз без поиска.
</div>

<script>
function setSelection() {
  var target = document.getElementById('ex5').getElementsByTagName('span')[0];
  var rng, sel;
  if ( document.createRange ) {
    rng = document.createRange();
    rng.selectNode( target )
    sel = window.getSelection();
    sel.removeAllRanges();
    sel.addRange( rng );
  } else {
    var rng = document.body.createTextRange();
    rng.moveToElementText( target );
    rng.select();
  }
}
</script>
```

В действии:

Снова будем выделять бабулю, на этот раз без поиска.

Выделить бабулю

Снятие выделения

Код для снятия выделения, использующий соответствующие методы объектов `Selection`:

```
function clearSelection() {
  try {
    // современный объект Selection
    window.getSelection().removeAllRanges();
  } catch(e) {
    // для IE<9
    document.selection.empty();
  }
}
```

Итого

- В современных браузерах поддерживается стандартный объект Range [28]
- В IE<9 поддерживается только собственный объект TextRange [29].

Есть библиотеки, которые «исправляют» объект `TextRange`, добавляя ему нужные свойства из `Range`.

Код, получающий выделение, при использовании такой библиотеки может выглядеть так:

```
var range = getRangeObject();
  if(range) {
    alert(range);
    alert(range.startContainer.nodeValue);
    alert(range.startOffset);
    alert(range.endOffset);
  } else {
    alert('Ничего не выделено');
  }
}
```

В действии:

Выделите текст:

The quick brown fox jumped over the lazy dog

Вывести выделение и свойства startContainer, startOffset, endOffset

Код функций `getRangeObject(win)` для получения выделения в окне и `fixIERangeObject(range, win)` для исправления `TextRange` — в песочнице вместе с этим примером: http://learn.javascript.ru/play/tutorial/browser/selection/fix-ie/index.html

Эта статья представляет собой обновлённый вариант статьи Александра Бурцева, сайта которого сейчас нет онлайн. Спасибо, Александр!

Применяем ООП: Drag'n'Drop++

Эта статья представляет собой продолжение статьи Drag’n’Drop объектов [30]. Она посвящена более гибкой и расширяемой реализации переноса.

В сложных приложениях Drag’n’Drop обладает рядом особенностей:

1. Перетаскиваются *элементы* из *зоны переноса* `dragZone` в *зону-цель* `dropTarget`. При этом сама зона не переносится.

   Например — два списка, нужен перенос элемента из одного в другой. В этом случае один список является зоной переноса, второй — зоной-целью.

   Возможно, что перенос осуществляется внутри одного и того же списка. При этом `dragZone == dropTarget`.

2. На странице может быть несколько разных зон переноса и зон-целей.
3. Обработка завершения переноса может быть асинхронной, с уведомлением сервера.
4. Должно быть легко добавить новый тип зоны переноса или зоны-цели, а также расширить поведение существующей.

5. Фреймворк для переноса должен быть расширяемым с учётом сложных сценариев.

Всё это вполне реализуемо. Но для этого фреймворк, описанный в статье Drag'n'Drop объектов [31], нужно отрефакторить, и разделить на сущности.

Основные сущности

Всего будет 4 сущности:

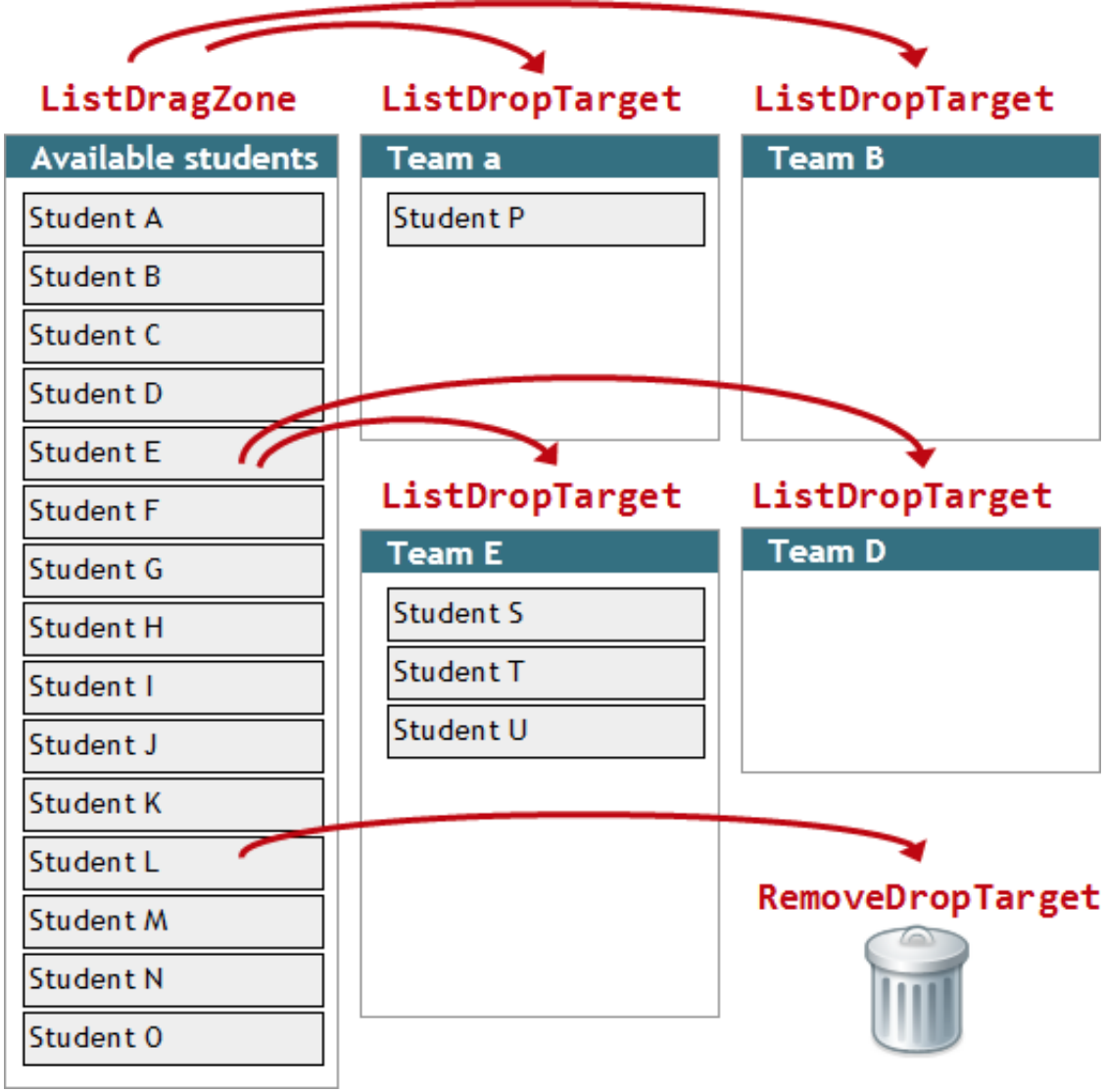
`DragZone`
: Зона переноса. С нее начинается перенос. Она принимает нажатие мыши и генерирует аватар нужного типа.

`DragAvatar`
: Переносимый объект. Предоставляет доступ к информации о том, что переносится. Умеет двигать себя по экрану. В зависимости от вида переноса, может что-то делать с собой в конце, например, самоуничтожаться.

`DropTarget`
: Зона-цель, на которую можно положить. В процессе переноса аватара над ней умеет рисовать на себе предполагаемое «место приземления». Обрабатывает окончание переноса.

`dragManager`
: Единый объект, который стоит над всеми ними, ставит обработчики `mousedown/mousemove/mouseup` и управляет процессом. В терминах ООП, это не класс, а объект-синглтон [32] , поэтому он с маленькой буквы.

На макете страницы ниже возможен перенос студентов из левого списка — вправо, в одну из команд или в «корзину»:



Здесь левый список является зоной переноса `ListDragZone`, а правые списки — это несколько зон-целей `ListDropTarget`. Кроме того, корзина также является зоной-целью отдельного типа `RemoveDropTarget`.

Пример

В этой статье мы реализуем пример, когда узлы дерева можно переносить внутри него. То есть, дерево, которое является одновременно `TreeDragZone` и `TreeDropTarget`.

Структура дерева будет состоять из вложенных списков с заголовком в `SPAN`:

```html
<ul>
  <li><span>Заголовок 1</span>
    <ul>
      <li><span>Заголовок 1.1</span></li>
      <li><span>Заголовок 1.2</span></li>
      ...
    </ul>
  </li>
  ...
</ul>
```

При переносе:

- Для аватара нужно клонировать заголовок узла, на котором было нажатие.
- Узлы, на которые можно положить, при переносе подсвечиваются красным.
- Нельзя перенести узел сам в себя или в своего потомка.
- Дерево само поддерживает сортировку по алфавиту среди узлов.
- Обязательна расширяемость кода, поддержка большого количества узлов и т.п.

Возьмите за любой заголовок и поменяйте ему родителя.
В собственных детей перенести нельзя.
Потомки всегда отсортированы по алфавиту.

- Древо жизни (сверхмалая часть)
  - Грибы
    - Древесные
      - Чага
    - Наземные
      - Опята
      - Подосиновики
  - Животные
    - Земноводные
      - Лягушки
      - Саламандры
      - Тритоны
    - Млекопитающие
      - Коровы
      - Ослы
      - Собаки
      - Тигры

dragManager

Обязанность `dragManager` — обработка событий мыши и координация всех остальных сущностей в процессе переноса.

Готовьтесь, дальше будет много кода с комментариями.

Следующий код должен быть очевиден по смыслу, если вы читали предыдущую статью [33]. Объект взят оттуда, и из него изъята лишняя функциональность, которая перенесена в другие сущности.

Если вызываемые в нём методы `onDrag*` непонятны — смотрите далее, в описание остальных объектов.

```
001 var dragManager = new function() {
002
003   var dragZone, avatar, dropTarget;
004   var downX, downY;
005
006   var self = this;
007
008   function onMouseDown(e){
009     e = fixEvent [34](e);
010
011     if (e.which != 1 ) { // не левой кнопкой
012       return false;
013     }
014
015     dragZone = findDragZone(e);
016
017     if (!dragZone) {
018         return;
019     }
020
021     // запомним, что элемент нажат на текущих координатах pageX/pageY
022     downX = e.pageX;
023     downY = e.pageY;
024
025     return false;
026   }
027
028   function onMouseMove(e) {
029     if (!dragZone) return; // элемент не зажат
030
031     e = fixEvent(e);
032
033     if ( !avatar ) { // элемент нажат, но пока не начали его двигать
034       if ( Math.abs(e.pageX-downX) < 3 && Math.abs(e.pageY-downY) < 3 ) {
035         return;
036       }
037       // попробовать захватить элемент
038       avatar = dragZone.onDragStart(downX, downY, e);
039
040       if (!avatar) { // не получилось, значит перенос продолжать нельзя
041         cleanUp(); // очистить приватные переменные, связанные с переносом
042         return;
043       }
044     }
045
```

```
    // отобразить перенос объекта, перевычислить текущий элемент под курсором
    avatar.onDragMove(e);

    // найти новый dropTarget под курсором: newDropTarget
    // текущий dropTarget остался от прошлого mousemove
    // *оба значения: и newDropTarget и dropTarget могут быть null
    var newDropTarget = findDropTarget(e);

    if (newDropTarget != dropTarget) {
      // уведомить старую и новую зоны-цели о том, что с них ушли/на них зашли
      dropTarget && dropTarget.onDragLeave(newDropTarget, avatar, e);
      newDropTarget && newDropTarget.onDragEnter(dropTarget, avatar, e);
    }

    dropTarget = newDropTarget;

    dropTarget && dropTarget.onDragMove(avatar, e);

    return false;
  }

  function onMouseUp(e) {
    e = fixEvent(e);

    if (e.which != 1 ) { // не левой кнопкой
      return false;
    }

    if (avatar) { // если уже начали передвигать

      if (dropTarget) {
        // завершить перенос и избавиться от аватара, если это нужно
        // эта функция обязана вызвать avatar.onDragEnd/onDragCancel
        dropTarget.onDragEnd(avatar, e);
      } else {
        avatar.onDragCancel();
      }

    }

    cleanUp();
  }

  function cleanUp() {
    // очистить все промежуточные объекты
    dragZone = avatar = dropTarget = null;
  }

  function findDragZone(event) {
    var elem = event.target;
    while(elem != document && !elem.dragZone) {
      elem = elem.parentNode;
    }
    return elem.dragZone;
  }

  function findDropTarget(event) {
```

```
    // получить элемент под аватаром
    var elem = avatar.getTargetElem();

    while(elem != document && !elem.dropTarget) {
      elem = elem.parentNode;
    }

    if (!elem.dropTarget) {
      return null;
    }

    return elem.dropTarget;
  }

  document.ondragstart = function() {
    return false;
  }

  document.onmousemove = onMouseMove;
  document.onmouseup = onMouseUp;
  document.onmousedown = onMouseDown;
};
```

DragZone

Основная задача `DragZone` — создать аватар и инициализовать его. В зависимости от места, где произошел клик, аватар получит соответствующий подэлемент зоны.

Метод для создания аватара `_makeAvatar` вынесен отдельно, чтобы его легко можно было переопределить и подставить собственный тип аватара.

```
/**
 * Зона, из которой можно переносить объекты
 * Умеет обрабатывать начало переноса на себе и создавать "аватар"
 * @param elem DOM-элемент, к которому привязана зона
 */
function DragZone(elem) {
  elem.dragZone = this;
  this._elem = elem;
}

/**
 * Создать аватар, соответствующий зоне.
 * У разных зон могут быть разные типы аватаров
 */
DragZone.prototype._makeAvatar = function() {
  /* override */
};

/**
 * Обработать начало переноса.
 *
 * Получает координаты изначального нажатия мышки, событие.
 *
 * @param downX Координата изначального нажатия по X
 * @param downY Координата изначального нажатия по Y
 * @param event текущее событие мыши
 *
 * @return аватар или false, если захватить с данной точки ничего нельзя
 */
DragZone.prototype.onDragStart = function(downX, downY, event) {

  var avatar = this._makeAvatar();

  if (!avatar.initFromEvent(downX, downY, event)) {
    return false;
  }

  return avatar;
};
```

TreeDragZone

Объект зоны переноса для дерева, по существу, не вносит ничего нового, по сравнению с `DragZone`.

Он только переопределяет `_makeAvatar` для создания `TreeDragAvatar`.

```
function TreeDragZone(elem) {
  DragZone.apply(this, arguments);
}

extend(TreeDragZone, DragZone);

TreeDragZone.prototype._makeAvatar = function() {
  return new TreeDragAvatar(this, this._elem);
};
```

DragAvatar

Аватар создается только зоной переноса при начале Drag'n'Drop. Он содержит всю необходимую информацию об объекте, который переносится.

В дальнейшем вся работа происходит *только с аватаром*, сама зона напрямую не вызывается.

У аватара есть три основных свойства:

`_dragZone`
: Зона переноса, которая его создала.

`_dragZoneElem`
: Элемент, соответствующий аватару в зоне переноса. По умолчанию — DOM-элемент всей зоны. Это подходит в тех случаях, когда зона перетаскивается только целиком.
При инициализации аватара значение этого свойства быть уточнено, например изменено на подэлемент списка, который перетаскивается.

`_elem`
: Основной элемент аватара, который будет двигаться по экрану. По умолчанию равен `_dragZoneElem`, т.е мы переносим сам элемент.

  При инициализации мы можем также склонировать `_dragZoneElem`, или создать своё красивое представление переносимого элемента и поместить его в `_elem`.

```
/**
 * "Аватар" - элемент, который перетаскивается.
 *
 * В простейшем случае аватаром является сам переносимый элемент
 * Также аватар может быть клонированным элементом
 * Также аватар может быть иконкой и вообще чем угодно.
 */
function DragAvatar(dragZone, dragElem) {
  /** "родительская" зона переноса */
  this._dragZone = dragZone;

  /**
   * подэлемент родительской зоны, к которому относится аватар
   * по умолчанию - элемент, соответствующий всей зоне
   * может быть уточнен в initFromEvent
   */
  this._dragZoneElem = dragElem;

  /**
   * Сам элемент аватара, который будет носиться по экрану.
   * Инициализуется в initFromEvent
   */
  this._elem = dragElem;
}

/**
 * Инициализовать this._elem и позиционировать его
 * При необходимости уточнить this._dragZoneElem
 * @param downX Координата X нажатия мыши
 * @param downY Координата Y нажатия мыши
 * @param event Текущее событие мыши
 */
```

```
DragAvatar.prototype.initFromEvent = function(downX, downY, event) {
  /* override */
};

/**
 * Возвращает информацию о переносимом элементе для DropTarget
 * @param event
 */
DragAvatar.prototype.getDragInfo = function(event) {
  // тут может быть еще какая-то информация, необходимая для обработки конца или процесса переноса
  return {
    elem: this._elem,
    dragZoneElem: this._dragZoneElem,
    dragZone: this._dragZone
  };
};

/**
 * Возвращает текущий самый глубокий DOM-элемент под this._elem
 * Приватное свойство _currentTargetElem обновляется при каждом передвижении
 */
DragAvatar.prototype.getTargetElem = function() {
  return this._currentTargetElem;
};

/**
 * При каждом движении мыши перемещает this._elem
 * и записывает текущий элемент под this._elem в _currentTargetElem
 * @param event
 */
DragAvatar.prototype.onDragMove = function(event) {
  this._elem.style.left = event.pageX - this._shiftX + 'px';
  this._elem.style.top = event.pageY - this._shiftY + 'px';

  this._currentTargetElem = getElementUnderClientXY(this._elem, event.clientX, event.clientY);
};

/**
 * Действия с аватаром, когда перенос не удался
 * Например, можно вернуть элемент обратно или уничтожить
 */
DragAvatar.prototype.onDragCancel = function() {
  /* override */
};

/**
 * Действия с аватаром после успешного переноса
 */
DragAvatar.prototype.onDragEnd = function() {
  /* override */
};
```

TreeDragAvatar

Основные изменения — в методе `initFromEvent`, который создает аватар из узла, на котором был клик.

Обратите внимание, возможно что клик был не на заголовке SPAN, а просто где-то на дереве. В этом случае initFromEvent возвращает false и перенос не начинается.

```
function TreeDragAvatar(dragZone, dragElem) {
  DragAvatar.apply(this, arguments);
}

extend(TreeDragAvatar, DragAvatar);

TreeDragAvatar.prototype.initFromEvent = function(downX, downY, event) {
  if (event.target.tagName != 'SPAN') return false;

  this._dragZoneElem = event.target;
  var elem = this._elem = this._dragZoneElem.cloneNode(true);
  elem.className = 'avatar';

  // создать вспомогательные свойства shiftX/shiftY
  var coords = getCoords(this._dragZoneElem);
  this._shiftX = downX - coords.left;
  this._shiftY = downY - coords.top;

  // инициировать начало переноса
  document.body.appendChild(elem);
  elem.style.zIndex = 9999;
  elem.style.position = 'absolute';

  return true;
};

/**
 * Вспомогательный метод
 */
TreeDragAvatar.prototype._destroy = function() {
  this._elem.parentNode.removeChild(this._elem);
};

/**
 * При любом исходе переноса элемент-клон больше не нужен
 */
TreeDragAvatar.prototype.onDragCancel = function() {
  this._destroy();
};

TreeDragAvatar.prototype.onDragEnd = function() {
  this._destroy();
};
```

DropTarget

Именно на DropTarget ложится работа по отображению предполагаемой «точки приземления» аватара, а также, по завершению переноса, обработка результата.

Как правило, DropTarget принимает переносимый узел в себя, а вот как конкретно организован процесс вставки — нужно описать в классе-наследнике. Разные типы зон делают разное при вставке: TreeDropTarget вставляет элемент в качестве потомка, а RemoveDropTarget — удаляет.

```
/**
 * Зона, в которую объекты можно класть
 * Занимается индикацией передвижения по себе, добавлением в себя
 */
function DropTarget(elem) {
  elem.dropTarget = this;
  this._elem = elem;

  /**
   * Подэлемент, над которым в настоящий момент находится аватар
   */
  this._targetElem = null;
}

/**
 * Возвращает DOM-подэлемент, над которым сейчас пролетает аватар
 *
 * @return DOM-элемент, на который можно положить или undefined
 */
DropTarget.prototype._getTargetElem = function(avatar, event) {
  return this._elem;
};

/**
 * Спрятать индикацию переноса
 * Вызывается, когда аватар уходит с текущего this._targetElem
 */
DropTarget.prototype._hideHoverIndication = function(avatar) {
  /* override */
};

/**
 * Показать индикацию переноса
 * Вызывается, когда аватар пришел на новый this._targetElem
 */
DropTarget.prototype._showHoverIndication = function(avatar) {
  /* override */
};

/**
 * Метод вызывается при каждом движении аватара
 */
DropTarget.prototype.onDragMove = function(avatar, event) {

  var newTargetElem = this._getTargetElem(avatar, event);

  if (this._targetElem != newTargetElem) {

    this._hideHoverIndication(avatar);
    this._targetElem = newTargetElem;
    this._showHoverIndication(avatar);
  }
};

/**
 * Завершение переноса.
 * Алгоритм обработки (переопределить функцию и написать в потомке):
```

```
 * 1. Получить данные переноса из avatar.getDragInfo()
 * 2. Определить, возможен ли перенос на _targetElem (если он есть)
 * 3. Вызвать avatar.onDragEnd() или avatar.onDragCancel()
 *  Если нужно подтвердить перенос запросом на сервер, то avatar.onDragEnd(),
 *  а затем асинхронно, если сервер вернул ошибку, avatar.onDragCancel()
 *  При этом аватар должен уметь "откатываться" после onDragEnd.
 *
 * При любом завершении этого метода нужно (делается ниже):
 *  снять текущую индикацию переноса
 *  обнулить this._targetElem
 */
DropTarget.prototype.onDragEnd = function(avatar, event) {
  this._hideHoverIndication(avatar);
  this._targetElem = null;
};

/**
 * Вход аватара в DropTarget
 */
DropTarget.prototype.onDragEnter = function(fromDropTarget, avatar, event) {
};

/**
 * Выход аватара из DropTarget
 */
DropTarget.prototype.onDragLeave = function(toDropTarget, avatar, event) {
  this._hideHoverIndication();
  this._targetElem = null;
};
```

Как видно, из кода выше, по умолчанию `DropTarget` занимается только отслеживанием и индикацией «точки приземления». По умолчанию, единственной возможной «точкой приземления» является сам элемент зоны. В более сложных ситуациях это может быть подэлемент.

Для применения в реальности необходимо как минимум переопределить обработку результата переноса в `onDragEnd`.

TreeDropTarget

`TreeDropTarget` содержит код, специфичный для дерева:

- Индикацию переноса над элементом: методы `_showHoverIndication` и `_hideHoverIndication`.
- Получение текущей точки приземления `_targetElem` в методе `_getTargetElem`. Ей может быть только заголовок узла дерева, причем дополнительно проверяется, что это не потомок переносимого узла.
- Обработка успешного переноса в `onDragEnd`, вставка исходного узла `avatar.dragZoneElem` в узел, соответствующий `_targetElem`.

```
function TreeDropTarget(elem) {
  TreeDropTarget.parent.constructor.apply(this, arguments);
}

extend(TreeDropTarget, DropTarget);

TreeDropTarget.prototype._showHoverIndication = function() {
  this._targetElem && addClass(this._targetElem, 'hover');
};
```

```
TreeDropTarget.prototype._hideHoverIndication = function() {
  this._targetElem && removeClass(this._targetElem, 'hover');
};

TreeDropTarget.prototype._getTargetElem = function(avatar, event) {
  var target = avatar.getTargetElem();
  if (target.tagName != 'SPAN') {
    return;
  }

  // проверить, может быть перенос узла внутрь самого себя или в себя?
  var elemToMove = avatar.getDragInfo(event).dragZoneElem.parentNode;

  var elem = target;
  while(elem) {
    if (elem == elemToMove) return; // попытка перенести родителя в потомка
    elem = elem.parentNode;
  }

  return target;
};

TreeDropTarget.prototype.onDragEnd = function(avatar, event) {

  if (!this._targetElem) {
    // перенос закончился вне подходящей точки приземления
    avatar.onDragCancel();
    return;
  }

  this._hideHoverIndication();

  // получить информацию об объекте переноса
  var avatarInfo = avatar.getDragInfo(event);

  avatar.onDragEnd(); // аватар больше не нужен, перенос успешен

  // вставить элемент в детей в отсортированном порядке
  var elemToMove = avatarInfo.dragZoneElem.parentNode; // <LI>
  var title = avatarInfo.dragZoneElem.innerHTML; // переносимый заголовок

  // получить контейнер для узлов дерева, соответствующий точке преземления
  var ul = this._targetElem.parentNode.getElementsByTagName('UL')[0];
  if (!ul) { // нет детей, создадим контейнер
    ul = document.createElement('UL');
    this._targetElem.parentNode.appendChild(ul);
  }

  // вставить новый узел в нужное место среди потомков, в алфавитном порядке
  var li = null;
  for(var i=0; i < ul.children.length; i++) {
    li = ul.children[i];
    var childTitle = li.children[0].innerHTML;
    if (childTitle > title) {
      break;
    }
```

```
  }

  ul.insertBefore(elemToMove, li);

  this._targetElem = null;
};
```

Итого

Реализация Drag'n'Drop оказалась отличным способом применить ООП в JavaScript.

Исходный код примера целиком находится здесь: http://learn.javascript.ru/play/tutorial/browser/dnd/dragTree/index.html.

- Синглтон `dragManager` и классы `Drag*` задают общий фреймворк. От них наследуются конкретные объекты. Для создания новых зон достаточно унаследовать стандартные классы и переопределить их.
- Мини-фреймворк для Drag'n'Drop, который здесь представлен, является переписанным и обновленным вариантом реальной библиотеки, на основе которой было создано много успешных скриптов переноса.

  В зависимости от ваших потребностей, вы можете расширить его, добавить перенос нескольких объектов одновременно, поддержку событий и другие возможности.
- На сегодняшний день в каждом серьезном фреймворке есть библиотека для Drag'n'Drop. Она работает похожим образом, но сделать универсальный перенос — штука непростая. Зачастую он перегружен лишним функционалом, либо наоборот — недостаточно расширяем в нужных местах.
  Понимание, как это все может быть устроено, на примере этой статьи, может помочь в адаптации существующего кода под ваши потребности.

Свойство dataSet для data-* атрибутов

Современные браузеры, кроме IE, поддерживают специальное DOM-свойство для работы с атрибутами, названия которых начинаются с `data-`.

Свойство `dataset` содержит все такие атрибуты. Причём названия вида `data-my-super-attr` становятся `dataset.mySuperAttr`:

```
<div data-widget-name="Menu">...</div>

<script>
  var div = document.body.children[0];
  alert( div.dataset.widgetName ); // Menu
</script>
```

Свойство `dataset` допускает также запись. Причем подсвойства, записываемые в него, автоматически синхронизируются с атрибутами:

```
<div>...</div>

<script>
  var div = document.body.children[0];

  // записать в свойство
  div.dataset.widgetClass = "value";

  // прочитать из атрибута
  alert( div.getAttribute('data-widget-class') ); // value
</script>
```

Эти свойства предназначены для добавления своей информации в HTML-разметку. Любые атрибуты, начинающиеся с `data-` являются валидными, согласно спецификации HTML5.

Куки, document.cookie

Для чтения и записи cookie используется свойство `document.cookie`. Однако, оно представляет собой не объект, а строку в специальном формате, для удобной манипуляций с которой нужны дополнительные функции.

Чтение `document.cookie`

Наверняка у вас есть cookie, которые привязаны к этому сайту. Давайте полюбуемся на них. Вот так:

```
alert( document.cookie );
```

Эта строка состоит из пар `ключ=значение`, которые перечисляются через точку с запятой с пробелом `; `.

Значит, чтобы прочитать cookie, достаточно разбить строку по `; `, и затем найти нужный ключ. Это можно делать либо через `split` и работу с массивом, либо через регулярное выражение.

Функция `getCookie(name)`

Следующая функция `getCookie(name)` возвращает cookie с именем `name`:

```
// возвращает cookie с именем name, если есть, если нет, то undefined
function getCookie(name) {
  var matches = document.cookie.match(new RegExp(
    "(?:^|; )" + name.replace(/([\.$?*|{}\(\)\[\]\\\/\+^])/g, '\\$1') + "=([^;]*)"
  ));
  return matches ? decodeURIComponent(matches[1]) : undefined;
}
```

Обратим внимание, что значение может быть любым. Если оно содержит символы, нарушающие форматирование, например, пробелы или `;`, то оно кодируется при помощи `encodeURIComponent`. Функция `getCookie` автоматически раскодирует его.

Запись в `document.cookie`

В `document.cookie` можно писать. При этом запись не перезаписывает существующие cookie, а дополняет к ним!

Например, такая строка поставит cookie с именем `userName` и значением `Vasya`:

```
document.cookie = "userName=Vasya";
```

…Однако, всё не так просто. У cookie есть ряд важных настроек, которые очень желательно указать, так как значения по умолчанию у них неудобны.

Эти настройки указываются после пары ключ=значение, каждое — после точки с запятой:

`path=/mypath`
: Путь, внутри которого будет доступ к cookie. Если не указать, то имеется в виду текущий путь и все пути ниже него.

  Как правило, используется `path=/`, то есть cookie доступно со всех страниц сайта.

`domain=site.com`
: Домен, на котором доступно cookie. Если не указать, то текущий домен. Допустимо указывать текущий домен `site.com` и его поддомены, например `forum.site.com`.

  Если указать специальную маску `.site.com`, то cookie будет доступно на сайте и всех его поддоменах. Это используется, например, в случаях, когда кука содержит данные авторизации и должна быть доступна как на `site.com`, так и на `forum.site.com`.

`expires=Tue, 19 Jan 2038 03:14:07 GMT`
: Дата истечения куки в формате GMT. Получить нужную дату можно, используя объект `Date`. Его можно установить в любое время, а потом вызвать `toUTCString()`, например:

  ```
  // +1 день от текущего момента
  var date = new Date;
  date.setDate( date.getDate() + 1 );
  alert( date.toUTCString() );
  ```

  Если дату не указать, то cookie будет считаться «сессионным». Такое cookie удаляется при закрытии браузера.
  Если дата в прошлом, то кука будет удалена.

`secure`
: Cookie можно передавать только по HTTPS.

Например, чтобы поставить cookie `name=value` по текущему пути с датой истечения через 60 секунд:

```
var date = new Date( new Date().getTime() + 60*1000 );
document.cookie="name=value; path=/; expires="+date.toUTCString();
```

Чтобы удалить это cookie:

```
var date = new Date(0);
document.cookie="name=; path=/; expires="+date.toUTCString();
```

При удалении значение не важно. Можно его не указывать, как сделано в коде выше.

Функция `setCookie(name, value, options)`

Если собрать все настройки воедино, вот такая функция ставит куки:

```
function setCookie(name, value, options) {
  options = options || {};

  var expires = options.expires;

  if (typeof expires == "number" && expires) {
    var d = new Date();
    d.setTime(d.getTime() + expires*1000);
    expires = options.expires = d;
  }
  if (expires && expires.toUTCString) {
    options.expires = expires.toUTCString();
  }

  value = encodeURIComponent(value);

  var updatedCookie = name + "=" + value;

  for(var propName in options) {
    updatedCookie += "; " + propName;
    var propValue = options[propName];
    if (propValue !== true) {
      updatedCookie += "=" + propValue;
     }
  }

  document.cookie = updatedCookie;
}
```

Аргументы:

name
: название cookie

value
: значение cookie (строка)

options
: Объект с дополнительными свойствами для установки cookie:

  expires
  : Время истечения cookie. Интерпретируется по-разному, в зависимости от типа:

    - Число — количество секунд до истечения. Например, `expires: 3600` — кука на час.
    - Объект типа `Date` — дата истечения.
    - Если expires в прошлом, то cookie будет удалено.
    - Если expires отсутствует или `0`, то cookie будет установлено как сессионное и исчезнет при закрытии браузера.

  path
  : Путь для cookie.

  domain
  : Домен для cookie.

  secure
  : Если `true`, то пересылать cookie только по защищенному соединению.

Функция `deleteCookie(name)`

Здесь всё просто — удаляем вызовом `setCookie` с датой в прошлом.

```
function deleteCookie(name) {
  setCookie(name, "", { expires: -1 })
}
```

Сторонние cookie

При работе с cookie есть важная тонкость, которая касается внешних ресурсов.

Теоретически, любой ресурс, который загружает браузер, может поставить cookie.

Например:

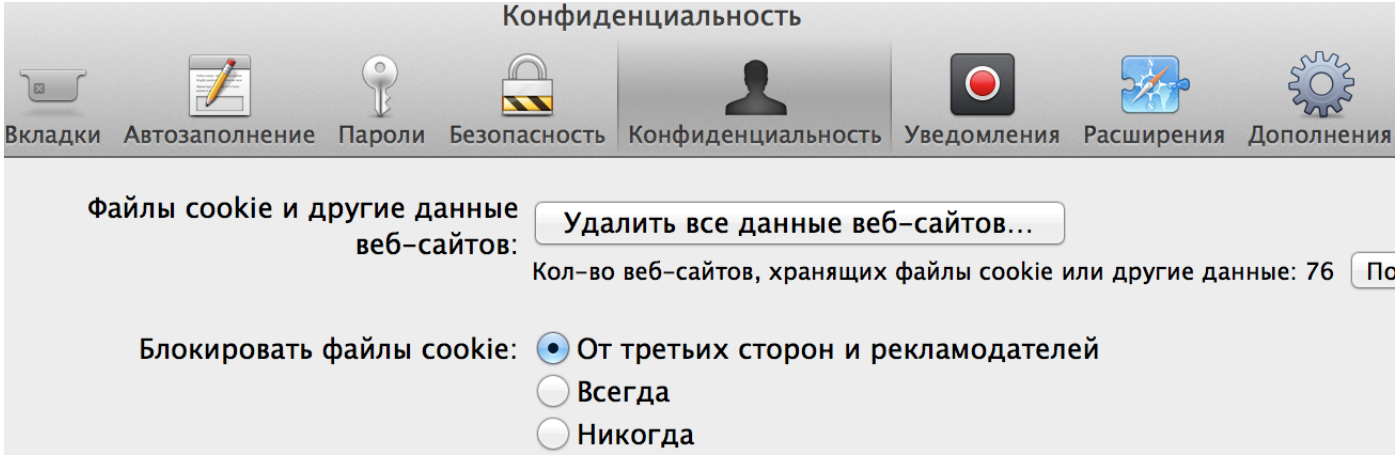
- Если на странице есть `<img src="http://mail.ru/counter.gif">`, то вместе с картинкой в ответ сервер может прислать заголовки, устанавливающие cookie.
- Если на странице есть `<iframe src="http://facebook.com/button.php">`, то во-первых сервер может вместе с `button.php` прислать cookie, а во-вторых JS-код внутри ифрейма может записать в `document.cookie`

При этом cookie будут принадлежать тому домену, который их поставил. То есть, на `mail.ru` для первого случая, и на `facebook.com` во втором.

Такие cookie, которые не принадлежат основной странице, называются «сторонними» (3rd party) cookies. Не все браузеры их разрешают.

Как правило, в настройках браузера можно поставить «Блокировать данные и файлы cookie сторонних сайтов» (Chrome).

В Safari такая настройка включена по умолчанию и выглядит так:



Тс-с-с. Большой брат смотрит за тобой.

Цель этого запрета — защитить посетителей от слежки со стороны рекламодателей, которые вместе с картинкой-баннером присылают и куки, таким образом помечая посетителей.

Например, на многих сайтах стоят баннеры и другая реклама Google Ads. При помощи таких cookie компания Google будет знать, какие именно сайты вы посещаете, сколько времени вы на них проводите и многое другое.

Как? Да очень просто — на каждом сайте загружается, к примеру, картинка с рекламой. При этом баннер берётся с домена,

принадлежащего Google. Вместе с баннером Google ставит cookie со специальным уникальным идентификатором.

Далее, при следующем запросе на баннер, браузер пошлёт стандартные заголовки, которые включают в себя:

- Cookie с домена баннера, то есть уникальный идентификатор, который был поставлен ранее.
- Стандартный заголовок Referrer (его не будет при HTTPS!), который говорит, с какого сайта сделан запрос. Да, впрочем, Google и так знает, с какого сайта запрос, ведь идентификатор сайта есть в URL.

Так что Google может хранить в своей базе, какие именно сайты из тех, на которых есть баннер Google, вы посещали, когда вы на них были, и т.п. Этот идентификатор легко привязывается к остальной информации от других сервисов, и таким образом картина слежки получается довольно-таки глобальной.

Здесь я не утверждаю, что в конкретной компании Google всё именно так… Но во-первых, сделать так легко, во-вторых идентификаторы действительно ставятся, а в-третьих, такие знания о человеке позволяют решать, какую именно рекламу и когда ему показать. А это основная доля доходов Google, благодаря которой корпорация существует.

Возможно, компания Apple, которая выпустила Safari, поставила такой флаг по умолчанию именно для уменьшения влияния Google?

А если очень надо?

Итак, Safari запрещает сторонние cookie по умолчанию. Другие браузеры предоставляют такую возможность, если посетитель захочет.

А что, если ну очень надо поставить стороннюю cookie, и чтобы это было надёжно?

Такая задача действительно возникает, например, в системе кросс-доменной авторизации, когда есть несколько доменов 2-го уровня, и хочется, чтобы посетитель, который входит в один сайт, автоматически распознавался во всей сетке. При этом cookie для авторизации ставятся на главный домен — «мастер», а остальные сайты запрашивают их при помощи специального скрипта (и, как правило, копируют к себе для оптимизации, но здесь это не суть).

Ещё пример — когда есть внешний виджет, например, `iframe` с информационным сервисом, который можно подключать на разные сайты. И этот `iframe` должен знать что-то о посетителе, опять же, авторизация или какие-то настройки, которые хорошо бы хранить в cookie.

Есть несколько способов поставить 3rd-party cookie для Safari.

Использовать ифрейм.
: Ифрейм является полноценным окном браузера. В нём должна быть доступна вся функциональность, в том числе cookie. Как браузер решает, что ифрейм «сторонний» и нужно запретить для него и его скриптов установку cookie? Критерий таков: «в ифрейме нет навигации». Если навигация есть, то ифрейм признаётся полноценным окном.

  Например, в сторонний `iframe` можно сделать POST. И тогда, в ответ на POST, сервер может поставить cookie. Или прислать документ, который это делает. Ифрейм, в который прошёл POST, считается родным и надёжным.

Popup-окно
: Другой вариант — использовать popup, то есть при помощи `window.open` открывать именно окно со стороннего домена, и уже там ставить cookie. Это тоже работает.

Редирект
: Ещё одно альтернативное решение, которое подходит не везде - это сделать интеграцию со сторонним доменом, такую что на него можно сделать редирект, он ставит cookie и делает редирект обратно.

Дополнительно

- На Cookie наложены ограничения:
  - Имя и значение (после `encodeURIComponent`) вместе не должны превышать 4кб.
  - Общее количество cookie на домен ограничено 30-50, в зависимости от браузера.

- Разные домены 2го уровня полностью изолированы. Но в пределах доменов 3го уровня куки можно ставить свободно с указанием `domain`.
- Сервер может поставить cookie с дополнительным флагом `HttpOnly`. Cookie с таким параметром передаётся только в заголовках, оно никак не доступно из JavaScript.

- Иногда посетители отключают cookie. Отловить это можно проверкой свойства navigator.cookieEnabled [35]

```
if (!navigator.cookieEnabled) {
  alert('Включите cookie для комфортной работы с этим сайтом');
}
```

…Конечно, предполагается, что включён JavaScript. Впрочем, посетитель без JS и cookie с большой вероятностью не человек, а бот.

Cookie.js

Файл с функциями для работы с cookie: cookie.js [36].

Решения задач

Решение задачи: Реализуйте drag'n'drop картинок в контейнер

Решение: http://learn.javascript.ru/play/tutorial/browser/dnd/dragImage/index.html

Нового кода в нем относительно немного, т.к. основная работа делается базовой библиотекой.

Весь новый код находится в файлах `ImageDragZone.js`, `ImageDragAvatar.js` и `ImageDropTarget.js`.

Ссылки

1. Dom.js http://learn.javascript.ru/files/tutorial/js/dom.js
2. AddClass/removeClass/hasClass(elem, cls) http://learn.javascript.ru/styles-and-classes#addClass
3. FixEvent(e, _this) http://learn.javascript.ru/fixevent#fixEvent
4. GetChar(event) http://learn.javascript.ru/keyboard-events#getChar
5. GetCoords(elem) http://learn.javascript.ru/coordinates#getCoords
6. QuerySelector http://learn.javascript.ru/searching-elements-dom#querySelector
7. QuerySelectorAll http://learn.javascript.ru/searching-elements-dom#querySelectorAll
8. Misc.js http://learn.javascript.ru/files/tutorial/js/misc.js
9. Bind(func, context) http://learn.javascript.ru/bind#bind
10. Описание http://learn.javascript.ru/bind#bind
11. Copy(dst, src1, src2…) http://learn.javascript.ru/arguments-pseudoarray#copy
12. Inherit(proto) http://learn.javascript.ru/prototype#inherit
13. Tmpl.js http://learn.javascript.ru/files/tutorial/js/tmpl.js
14. Tmpl(str) http://learn.javascript.ru/templates#tmpl
15. Шаблонизация в JavaScript http://learn.javascript.ru/templates
16. Animate http://learn.javascript.ru/js-animation#animate
17. Animate.js http://learn.javascript.ru/files/tutorial/js/animate.js

18. JS-Анимация http://learn.javascript.ru/js-animation
19. DOM Range http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges.html
20. Ссылки
21. Объект TextRange http://msdn.microsoft.com/en-us/library/ms535872.aspx
22. Источник http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges.html#td-boundarypoint
23. Свойства и методы Range http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges.html#Level-2-Range-Interface
24. Свойства и методы объекта TextRange http://msdn.microsoft.com/en-us/library/ms535872.aspx
25. Вряд ли понадобятся http://msdn.microsoft.com/en-us/library/ms536422.aspx
26. Метода execCommand http://msdn.microsoft.com/en-us/library/ms536419(v=vs.85).aspx
27. В примере с миксом #demo-mix
28. Range http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges.html
29. TextRange http://help.dottoro.com/ljgbbkjf.php
30. Drag’n’Drop объектов http://learn.javascript.ru/drag-and-drop-objects
31. Drag’n’Drop объектов http://learn.javascript.ru/drag-and-drop-objects
32. Объект-синглтон http://ru.wikipedia.org/wiki/Одиночка_(шаблон_проектирования)
33. Предыдущую статью http://learn.javascript.ru/drag-and-drop-objects
34. FixEvent http://learn.javascript.ru/mouse-events#fixEvent
35. Navigator.cookieEnabled https://developer.mozilla.org/en-US/docs/DOM/window.navigator.cookieEnabled
36. Cookie.js http://learn.javascript.ru/files/tutorial/browser/cookie/cookie.js