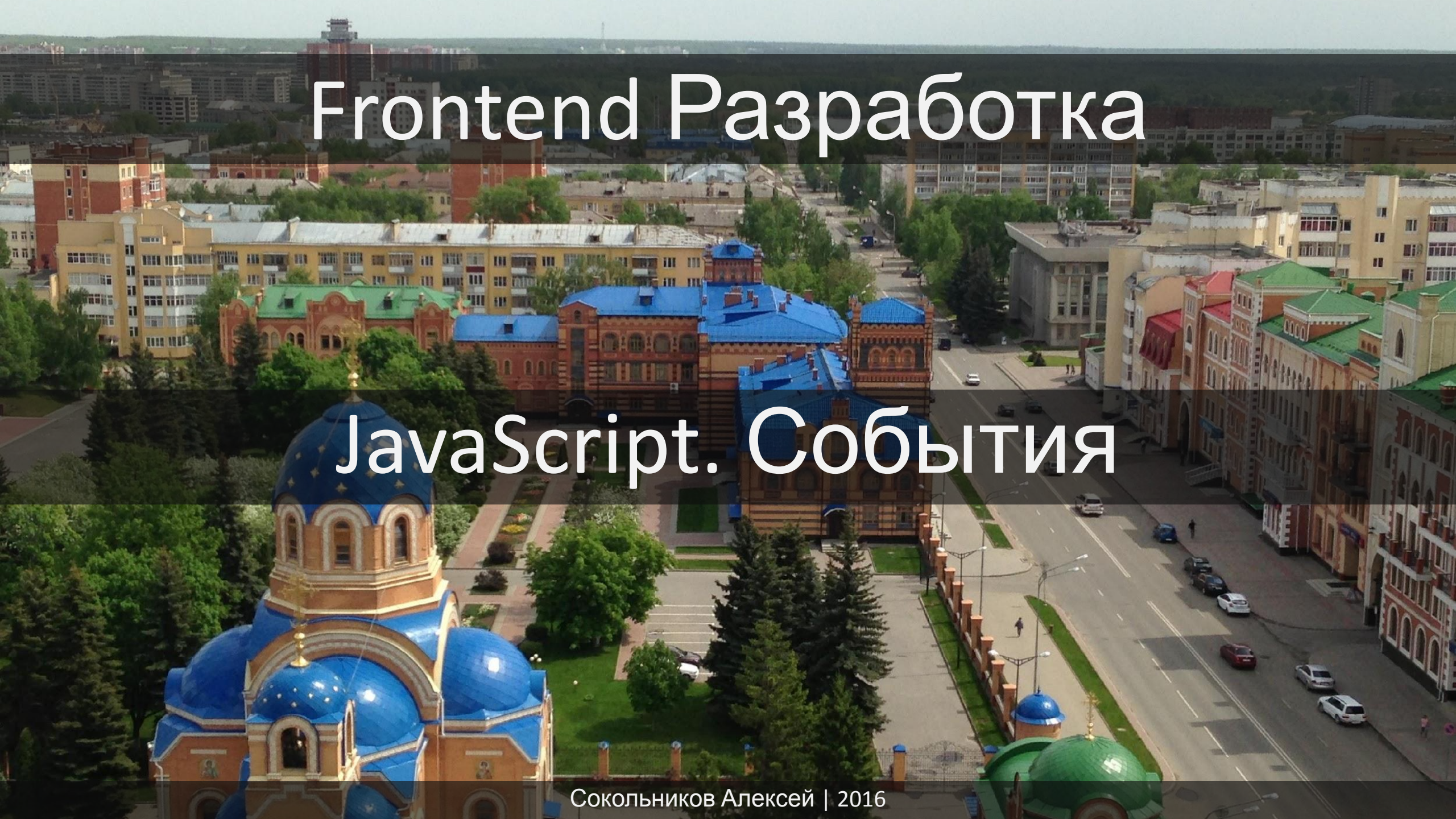


Frontend Разработка

JavaScript. События

Сокольников Алексей | 2016

События мыши:

- click – происходит, когда кликнули на элемент левой кнопкой мыши
- contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
- mouseover – возникает, когда на элемент наводится мышь
- mousedown и mouseup – когда кнопку мыши нажали или отжали
- mousemove – при движении мыши

События на элементах управления:

- submit – посетитель отправил форму <form>
- focus – посетитель фокусируется на элементе, например нажимает на <input>

Клавиатурные события:

- keydown – когда посетитель нажимает клавишу
- keyup – когда посетитель отпускает клавишу

События документа:

- DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

События CSS:

- transitionend – когда CSS-анимация завершена.

Назначение обработчиков событий

Использование атрибута HTML

```
<input value="Нажми меня" onclick="alert('Клик!')" type="button">
```

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <script>
     function countRabbits()  {
        for(var i=1; i<=3; i++) {
           alert("Кролик номер " + i);
        }
     }
  </script>
 </head>
 <body>
  <input type="button" onclick="countRabbits()" value="Считать кроликов!"/>
 </body>
</html>
```

Использование свойства DOM-объекта

```
<input id="elem" type="button" value="Нажми меня" />
<script>
   elem.onclick = function()  {
      alert( 'Спасибо' );
   };
</script>
```

Функция по событию

```
function sayThanks()  {
    alert( 'Спасибо!' );
}
elem.onclick = sayThanks;
```

Доступ к элементу через this

```
<button onclick="alert(this.innerHTML)">Нажми меня</button>
```

Недостаток назначения через свойство

```
elem.onclick = function()  {
    alert( ‘Пятница, 18-50' );
};

elem.onclick = function()  {
    alert( ‘Добрый вечер' );
};

elem.onclick = function()  {
    alert( ‘Лекция по Frontend’ );
}; ботчик
```

addEventListener

```
element.addEventListener(event, handler[, phase]);
```

- event - имя события, например click
- handler - ссылка на функцию, которую надо поставить обработчиком
- phase - необязательный аргумент, «фаза», на которой обработчик должен сработать.

removeEventListener

```
// передать те же аргументы, что были у addEventListener
element.removeEventListener(event, handler[, phase]);
```

Удаление требует именно ту же функцию

```
elem.addEventListener( "click" , function()  {
    alert('Спасибо!')
});
// ....
elem.removeEventListener( "click", function()  {
alert('Спасибо!')
});
```

```
function ThankYou()  {
    alert( 'Спасибо!' );
}

input.addEventListener("click", ThankYou);
// ....
input.removeEventListener("click", ThankYou);
```

```
<input id="elem" type="button" value="Нажми меня“ />
<script>
   function handler1()  {
      alert('Спасибо!');
   };
   function handler2()  {
      alert('Спасибо ещё раз!');
   }
   elem.onclick = function()  {
      alert("Привет");
   };
   elem.addEventListener("click", handler1); // Спасибо!
   elem.addEventListener("click", handler2); // Спасибо ещё раз!
</script>
```

addEventListener работает всегда, а DOM-свойство – нет

Порядок обработки событий

Главный поток

- В каждом окне выполняется только **один *главный* поток**, который занимается выполнением JavaScript, отрисовкой и работой с DOM.

- Он выполняет команды **последовательно**, может делать только одно дело одновременно и блокируется при выводе модальных окон, таких как alert.

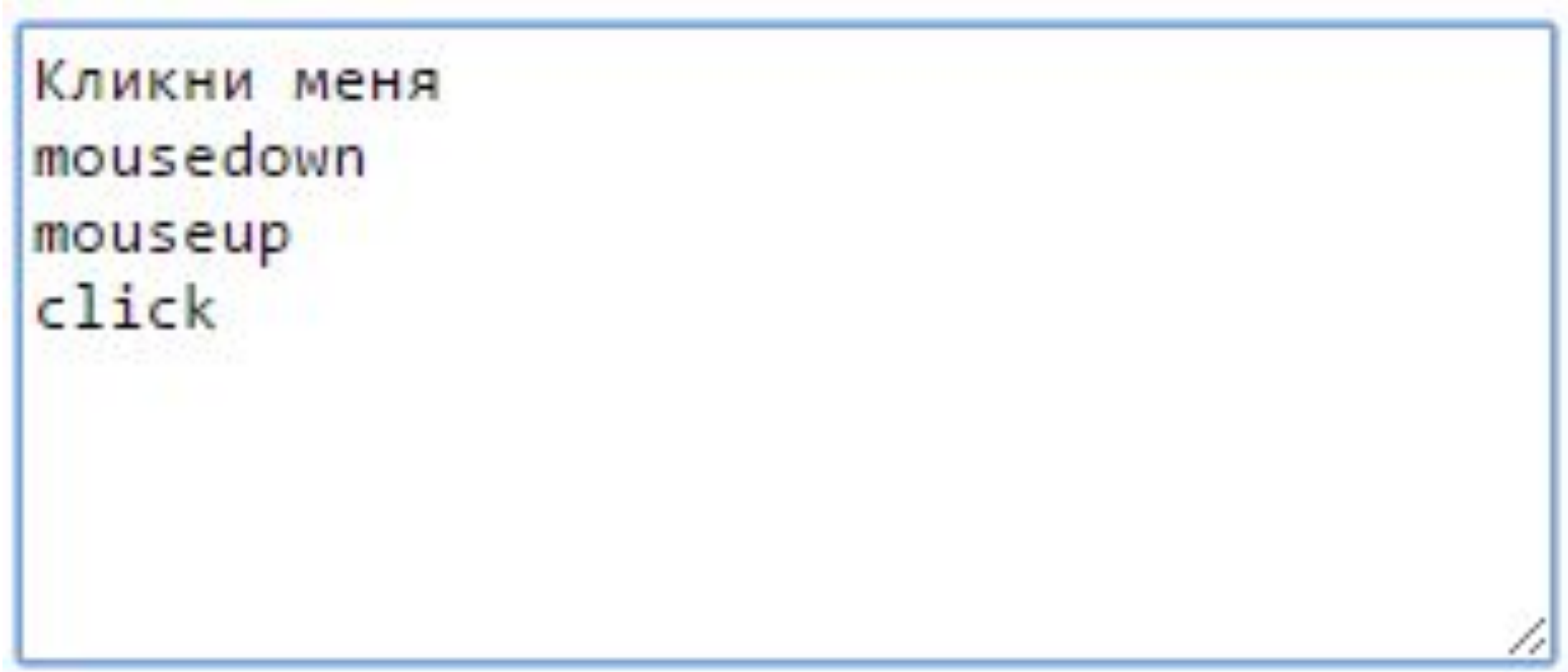
- **Дополнительные потоки**
- Web Workers

Очередь событий

- Когда происходит событие, оно попадает в очередь.
- Иногда события добавляются в очередь сразу пачкой.

```
<textarea rows="8" cols="40" id="area">Кликни меня </textarea>
<script>
   area.onmousedown = function(event)  {
      this.value += "mousedown\n";
      this.scrollTop = this.scrollHeight;
   };
   area.onmouseup = function(event)  {
      this.value += "mouseup\n";
      this.scrollTop = this.scrollHeight;
   };
   area.onclick = function(event) {
      this.value += "click\n";
      this.scrollTop = this.scrollHeight;
   };
</script>
```

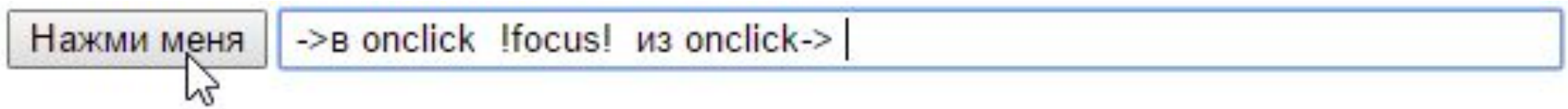
Результат



Вложенные (синхронные) события

```
<input type="button" id="button" value="Нажми меня" />
<input type="text" id="text" size="60" />
<script>
    button.onclick = function()  {
        text.value += ' ->в onclick ';
        text.focus(); // вызов инициирует событие onfocus
        text.value += ' из onclick-> ';
    };
    text.onfocus = function()  {
        text.value += ' !focus! ';
    };
</script>
```

Результат



Исключение в IE

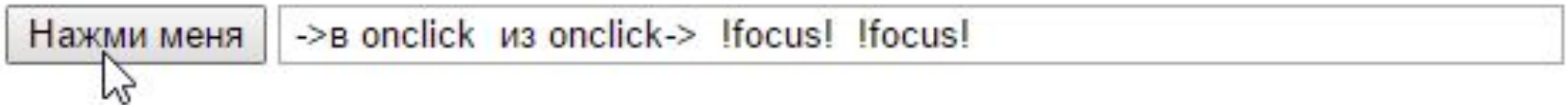
Так ведут себя все браузеры, кроме IE.

В нём событие onfocus – всегда асинхронное, так что будет сначала полностью обработан клик, а потом – фокус. В остальных – фокус вызовется посередине клика.

Делаем события асинхронными через setTimeout(…,0)

```
<input type="button" id="button" value="Нажми меня" />
<input type="text" id="text" size="60" />
<script>
   button.onclick = function()  {
      text.value += ' ->в onclick ';
      setTimeout(function()  {
         text.focus(); // сработает после onclick
      }, 0);
      text.value += ' из onclick-> ';
   };
   text.onfocus = function()  {
      text.value += ' !focus! ';
   };
</script>
```

Результат



Объект события

Свойства объекта события

```
<input type="button" value="Нажми меня" id="elem" />
<script>
  elem.onclick = function(event) {
    // вывести тип события, элемент и координаты клика
    alert(event.type + " на " + event.currentTarget);
    alert(event.clientX + ":" + event.clientY);
  }
</script>
```

Свойства объекта event:

- event.type - тип события, в данном случае click
- event.currentTarget - элемент, на котором сработал обработчик. Значение – в точности такое же, как и у this, но бывают ситуации, когда обработчик является методом объекта и его this при помощи bind привязан к этому объекту, тогда мы можем использовать event.currentTarget
- event.clientX / event.clientY - координаты курсора в момент клика (относительно окна)

Объект события доступен и в HTML

```
<input type="button" onclick="alert(event.type)" value="Тип события" />
```

jQuery

Функция $()

Базовые селекторы

- "*" – все элементы
- ".className" - элементы с классом *className*
- "#idName" - элемент с идентификатором *idName*
- "tagName" - элементы с заданным именем тега

Селекторы по атрибутам

- "[name]" - элементы, содержащие атрибут *name*
- "[name = value]" - элементы, у которых значение атрибута *name* совпадает с *value*
- "[name != value]" – элементы, у которых значение атрибута *name* не совпадает с *value*
- "[name ^= value]" – элементы, у которых значение атрибута *name* начинается с *value*
- и другие

Простые фильтры

- ":first" - первый найденный элемент
- ":even" - элементы с четными номерами позиций, в наборе выбранных элементов
- ":hidden" - невидимые элементы страницы
- и другие

Например

- $("div") - вернет все div-элементы на странице.
- $(".someBlock") - вернет все элементы с классом someBlock.
- $("#content") - вернет элемент с идентификатором content.
- $("#content2 div.someBlock") - вернет div-элементы с классом someBlock, которые находятся внутри элемента с идентификатором content2.
- $("div:odd") - вернет div-элементы, находящиеся на странице под нечетными номерами.
- $("[value = 5]") - вернет все элементы с атрибутом value, равным 5.

Манипуляции

- $("#bigIt").css("height") - возвратит значение высоты у элемента с идентификатором bigIt.
- $("div").css("width", "20px") - установит новое значение ширины всем div-элемента на странице.
- $("#bigIt").attr("class") - возвратит значение класса элемента с id = bigIt.
- $("#bigIt").html(Новье!) - изменит все html-содержимое элемента с id = bigIt, на заданное в методе html.
- $("#bigIt").text() - возвратит текст, находящийся внутри элемента с id = bigIt.
- $(".someBox").empty() - очистить от содержимого элементы с классом someBox.

Цепочки методов

```
$("#bigIt").empty().attr("class“, "noContent");
// в результате, у элемента с идентификатором bigIt будет удалено все содержимое,
// после чего ему будет установлен класс noContent.
```

Работа с набором элементов

- $("div").parent() - вернет родительские элементы всех div-ов.
- $("div").children() - вернет дочерние элементы всех div-ов.
- $("#someId").next() - вернет элемент, лежащий сразу после someId.
- $("div").prev() - вернет элементы, лежащие перед div'ами.
- $("div").eq(i) - вернет div-элемент, с индексом i в наборе.
- $("div").get(i) - вернет DOM-объект div'а, с индексом i.
- $("div").get() - вернет массив DOM-объеков всех div-ов.
- $("div").size() - вернет размер набора (количествово div-ов).

.each()

```
var heights = []; // переменная, которая будет хранить высоты элементов
$("div").each(function(index, element) {
    heights.push($(element).height());
});

// в итоге, в переменную heights будут помещены значения высот всех div-элементов
```