# Формы, элементы управления

## Навигация

Формы в документе входят в специальную коллекцию **document.forms**. Это так называемая «именованная» коллекция: мы можем использовать для получения формы как её **имя**, так и **порядковый номер** в документе.

Когда мы уже получили форму, любой элемент доступен в именованной коллекции form**.elements**. Например:

```
<form name="my">
  <input name="one" value="1">
  <input name="two" value="2">
</form>

<script>
  // получаем форму
  let form = document.forms.my; // <form name="my"> element

  // получаем элемент
  let elem = form.elements.one; // <input name="one"> element

  alert(elem.value); // 1
</script>
```

Может быть несколько элементов с одним и тем же именем, это часто бывает с кнопками-переключателями radio. В этом случае form.elements[name] является коллекцией

Форма может содержать один или несколько элементов <fieldset> внутри себя. Они также поддерживают свойство elements, в котором находятся элементы управления внутри них.

Есть более короткая запись: мы можем получить доступ к элементу через form[**index/name].**
Другими словами, вместо **form.elements.login** мы можем написать form.login.

Для любого элемента форма доступна также обратная ссылка через element.**form**. Так что форма ссылается на все элементы, а эти элементы ссылаются на форму.

*Значения элементов формы*:

- **input и textarea**: доступны через input.value, textarea.value, input.checked для чекбоксов и переключателей. Обратим внимание: хоть элемент **<textarea>**...</textarea> и хранит своё значение как вложенный HTML, нам не следует использовать textarea.innerHTML для доступа к нему. Там хранится только тот HTML, который **был изначально на странице**, а не текущее значение.
- **select и option**:
  1. select.**options** – коллекция из подэлементов `<option>`,
  2. select.**value** – значение выбранного в данный момент `<option>`,
  3. select.**selectedIndex** – номер выбранного `<option>`
  4.

  Они дают три разных способа установить значение в <select>:
  1. Найти соответствующий элемент <option> и установить в option.selected значение true (в случае multiple использовать только этот способ).
  2. Установить в select.value значение нужного <option>.
  3. Установить в select.selectedIndex номер нужного <option>.

  Полное описание элемента <select> доступно в спецификации https://html.spec.whatwg.org/multipage/forms.html#the-select-element.

  **Элемент <option>** редко используется сам по себе, но и здесь есть кое-что интересное. В спецификации есть красивый короткий синтаксис для создания элемента <option>:

  option = **new Option**(text, value, defaultSelected, selected);

  Параметры:
  *text* – текст внутри <option>,
  *value* – значение,
  *defaultSelected* – если true, то ставится HTML-атрибут selected,
  *selected* – если true, то элемент <option> будет выбранным.

- **Элементы <option>** имеют свойства:
  option.**selected** - Выбрана ли опция.
  option.**index** - Номер опции среди других в списке <select>.

option.**value** - Значение опции.
option.**text** - Содержимое опции (то, что видит посетитель).

## focus/blur

**Событие** focus вызывается в момент фокусировки, а blur – когда элемент теряет фокус. Их используют для валидации(проверки) введённых данных.

Современный HTML позволяет делать валидацию с помощью атрибутов **required**, **pattern** и т.д. Также существует HTML-атрибут **autofocus**, который устанавливает фокус на элемент, когда страница загружается.

События focus и blur **не всплывают**, но передаются вниз на фазе перехвата.

События focusin и focusout – такие же, как и focus/blur, но они всплывают.

**Методы** elem.**focus**() и elem.**blur**() устанавливают/снимают фокус. Например, запретим посетителю переключаться с поля ввода, если введённое значение не прошло валидацию. Это сработает во всех браузерах, **кроме Firefox** (bug).

Отметим, что мы **не можем** «отменить потерю фокуса», вызвав **event.preventDefault()** в обработчике onblur потому, что onblur срабатывает после потери фокуса элементом.

Потеря фокуса может произойти по множеству причин. Одна из них – когда посетитель кликает куда-то ещё. Но и JavaScript может быть причиной, например: alert переводит фокус на себя – элемент теряет фокус (событие blur), а когда alert закрывается – элемент получает фокус обратно (событие focus).

Многие элементы по умолчанию не поддерживают фокусировку. Какие именно – зависит от браузера, но одно всегда верно: поддержка focus/blur гарантирована для элементов, с которыми посетитель может взаимодействовать: <button>, <input>, <select>, <a> и т.д. С другой стороны, элементы форматирования , , <table> – по умолчанию не могут получить фокус. Метод elem.focus() не работает для них, и события focus/blur никогда не срабатывают.

Это можно изменить HTML-атрибутом tabindex. Любой элемент поддерживает фокусировку, если имеет tabindex. Значение этого атрибута – порядковый номер элемента, когда клавиша Tab (или что-то аналогичное) используется для переключения между элементами.

Порядок перебора таков:

- сначала идут элементы со значениями tabindex от 1 и выше, в порядке tabindex, а затем элементы без tabindex (например, обычный <input>).
- **tabindex="0"** ставит элемент в один ряд с элементами без tabindex. Обычно используется, чтобы включить фокусировку на элементе, но не менять порядок переключения. Чтобы элемент мог участвовать в форме наравне с обычными <input>.
- **tabindex="-1"** позволяет фокусироваться на элементе только программно.

## Событие: change

Событие change срабатывает по окончании изменения элемента. Для текстовых <input> это означает, что событие происходит при потере фокуса. Для других элементов: select, input type=checkbox/radio событие запускается сразу после изменения значения.

## Событие: input

Событие input срабатывает каждый раз при изменении значения. В отличие от событий клавиатуры, оно работает при любых изменениях значений, даже если они не связаны с клавиатурными действиями: вставка с помощью мыши или распознавание речи при диктовке текста. Если мы хотим обрабатывать каждое изменение в <input>, то это событие является лучшим выбором. Нажатия клавиш ⇦, ⇨ и подобных при фокусе на текстовом поле не вызовут это событие.

Событие input происходит после изменения значения. Поэтому мы **не можем использовать event.preventDefault()**

## События: cut, copy, paste

Эти события происходят при вырезании/копировании/вставке данных. Они относятся к классу ClipboardEvent и обеспечивают доступ к копируемым/вставляемым данным. Мы также **можем использовать event.preventDefault()** для предотвращения действия по умолчанию, и в итоге ничего не скопируется/не вставится.
Также запрещается генерировать «пользовательские» события буфера обмена при помощи dispatchEvent во всех браузерах, кроме Firefox.

## Событие: submit

Есть два основных способа отправить форму:

- Первый – нажать кнопку <input type="**submit**"> или <input type="**image**">.
- Второй – нажать **Enter**, находясь на каком-нибудь поле.

Оба действия сгенерируют событие submit на форме. Обработчик может проверить данные, и если есть ошибки, показать их и вызвать **event.preventDefault(),** тогда форма не будет отправлена на сервер.

**Взаимосвязь между submit и click**
При отправке формы по нажатию Enter в текстовом поле, генерируется сначала событие **click** на кнопке <input type="submit">.
Это довольно забавно, учитывая что никакого клика не было.

## Метод: submit

Чтобы отправить форму на сервер вручную, мы можем вызвать метод form.**submit**().
При этом событие submit не генерируется. Иногда это используют для генерации формы и отправки её вручную