Формы. JS.

Навигация: формы и элементы

Формы в документе входят в специальную коллекцию **document.forms**.

```
document.forms.my - форма с именем "my" (name="my")
document.forms[0] - первая форма в документе
```

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

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

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 через **element.form**. Элементы хранят ссылку на свою форму в свойстве **form**. Так что форма ссылается на все элементы, а эти элементы ссылаются на форму.

input и textarea

К их значению можно получить доступ через свойство **input.value** **(строка)** или **input.checked** (булево значение) для чекбоксов. Вот так:

```
input.value = "Новое значение";
textarea.value = "Новый текст";
input.checked = true; // для чекбоксов и переключателей
```

! Используйте **textarea.value** вместо **textarea.innerHTML**

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

Элемент <select> имеет 3 важных свойства:
select.options – коллекция из подэлементов <option>,
select.value – значение выбранного в данный момент <option>,
select.selectedIndex – номер выбранного <option>

Они дают три разных способа установить значение в <select>:

- Найти соответствующий элемент <option> и установить в option.selected значение true.
- Установить в select.value значение нужного <option>.
- Установить в select.selectedIndex номер нужного <option>

```
<select id="select">
  <option value="apple">Яблоко</option>
  <option value="pear">Груша</option>
  <option value="banana">Банан</option>
</select>
 // все три строки делают одно и то же
  select.options[2].selected = true;
  select.selectedIndex = 2;
  select.value = 'banana';
```

В отличие от большинства других элементов управления, <select> позволяет нам выбрать несколько вариантов одновременно, если у него стоит атрибут **multiple**. Эту возможность используют редко, но в этом случае для работы со значениями необходимо использовать первый способ, то есть ставить или удалять свойство selected у подэлементов <option>.Их коллекцию можно получить как **select.options**

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.

Тут может быть небольшая путаница с defaultSelected и selected. Всё просто: defaultSelected задаёт HTML-атрибут, его можно получить как option.getAttribute('selected'), а selected – выбрано значение или нет, именно его важно поставить правильно.
Н-р

```
let option = new Option("Текст", "value", true, true);
```

Фокусировка: focus/blur

Элемент получает фокус (событие **focus**), когда пользователь кликает по нему или использует клавишу Tab. Фокусировка обычно означает: «приготовься к вводу данных на этом элементе». Момент потери фокуса (**blur**) - это момент, когда пользователь кликает куда-то ещё или нажимает Tab, чтобы переключиться на следующее поле формы. Потеря фокуса обычно означает «данные введены», и мы можем выполнить проверку введённых данных или даже отправить эти данные на сервер и так далее. Н-р, проверка формы:

```
input.onblur = function() {
  if (!input.value.includes('@')) { // не email
    input.classList.add('invalid');
    error.innerHTML = 'Пожалуйста, введите правильный email.'
  }
};
input.onfocus = function() {
  if (this.classList.contains('invalid')) {
    // удаляем индикатор ошибки, т.к. пользователь хочет ввести данные заново
    this.classList.remove('invalid');
    error.innerHTML = "";
  }
};
```

Методы **elem.focus() и elem.blur()** устанавливают/снимают фокус. Отметим, что мы не можем «отменить потерю фокуса», вызвав event.preventDefault() в обработчике onblur потому, что onblur срабатывает после потери фокуса элементом.

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. То есть: если у нас два элемента, первый имеет tabindex="1", а второй tabindex="2", то находясь в первом элементе и нажав Tab – мы переместимся во второй. Порядок перебора таков: сначала идут элементы со значениями tabindex от 1 и выше, в порядке tabindex, а затем элементы без tabindex (например, обычный <input>). При совпадающих tabindex элементы перебираются в том порядке, в котором идут в документе. Есть два специальных значения:

- **tabindex="0**" ставит элемент в один ряд с элементами без tabindex. То есть, при переключении такие элементы будут после элементов с tabindex ≥ 1. 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. Клавиша Tab проигнорирует такой элемент, но метод elem.focus() будет действовать.

Мы можем добавить tabindex из JavaScript, используя свойство **elem.tabIndex.**

События focusin/focusout

События **focus** и **blur не всплывают**. Например, мы не можем использовать onfocus на <form>, чтобы подсветить её. У этой проблемы два решения:

Первое: забавная особенность – focus/blur не всплывают, но передаются вниз на фазе перехвата.
Это сработает:

```
<form id="form">
  <input type="text" name="name" value="Имя">
  <input type="text" name="surname" value="Фамилия">
</form>
<style> .focused { outline: 1px solid red; } </style>

<script>
  // установить обработчик на фазе перехвата (последний аргумент true)
  form.addEventListener("focus", () => form.classList.add('focused'), true);
  form.addEventListener("blur", () => form.classList.remove('focused'), true);
</script>
```

Второе решение: события **focusin** и **focusout** – такие же, как и focus/blur, но они **всплывают**.
Заметьте, что эти события должны использоваться с elem.addEventListener, но не с on<event>.
Второй рабочий вариант:

```
<form id="form">
  <input type="text" name="name" value="Имя">
  <input type="text" name="surname" value="Фамилия">
</form>
<style> .focused { outline: 1px solid red; } </style>

<script>
  form.addEventListener("focusin", () => form.classList.add('focused'));
  form.addEventListener("focusout", () => form.classList.remove('focused'));
</script>
```

Текущий элемент с фокусом можно получить из **document.activeElement**.

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. Событие input не происходит при вводе с клавиатуры или иных действиях, если при этом не меняется значение в текстовом поле, т.е. нажатия клавиш ⇦, ⇨ и подобных при фокусе на текстовом поле не вызовут это событие. **event.preventDefault()** для input **не работает**.

События: cut, copy, paste

Эти события происходят при вырезании/копировании/вставке данных. Они относятся к классу ClipboardEvent и обеспечивают доступ к копируемым/вставляемым данным. Свойство event.clipboardData предоставляет доступ на чтение/запись в буфер обмена…Мы также можем использовать **event.preventDefault()** для предотвращения действия по умолчанию, и в итоге ничего не скопируется/не вставится. Например, код, приведённый ниже, предотвращает все подобные события и показывает, что мы пытаемся вырезать/копировать/вставить:

```
<input type="text" id="input">
<script>
  input.oncut = input.oncopy = input.onpaste = function(event) {
    alert(event.type + ' - ' + event.clipboardData.getData('text/plain'));
    return false;
  };
```

```
</script>
```

Событие: submit

При отправке формы срабатывает событие **submit**, оно обычно используется для проверки (валидации) формы перед её отправкой на сервер или для предотвращения отправки и обработки её с помощью JavaScript. Метод form.submit() позволяет инициировать отправку формы из JavaScript. Мы можем использовать его для динамического создания и отправки наших собственных форм на сервер.

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.

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

Чтобы отправить форму на сервер вручную, мы можем вызвать метод **form.submit().** При этом событие submit не генерируется. Предполагается, что если программист вызывает метод form.submit(), то он уже выполнил всю соответствующую обработку. Иногда это используют для генерации формы и отправки её вручную, например так:

```
let form = document.createElement('form');
form.action = 'https://google.com/search';
form.method = 'GET';
form.innerHTML = '<input name="q" value="test">';

// перед отправкой формы, её нужно вставить в документ
document.body.append(form);

form.submit();
```

Событие: reset

Очистка формы. Восстанавливает стандартные значения всем элементам формы. Данный метод выполняет действие идентичное нажатию кнопки имеющей тип **reset**.