JavaScript - формы

Свойства и методы формы, Фокусировка: focus/blur,
События: change, input, cut, copy, paste,
Отправка формы: событие и метод submit

Свойства и методы формы

- Навигация: формы и элементы
- Обратная ссылка: element.form
- Элементы формы:
  - input и textarea
  - select и option
  - new Option

Навигация: формы и элементы

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.

Может быть несколько элементов с одним и тем же именем, это часто бывает с кнопками-переключателями radio.

Эти навигационные свойства не зависят от структуры тегов внутри формы. Все элементы управления формы, как бы глубоко они не находились в форме, доступны в коллекции form.elements.

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

```
<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>
```

```
<form>
  <input type="radio" name="age" value="10">
  <input type="radio" name="age" value="20">
</form>

<script>
let form = document.forms[0];

let ageElems = form.elements.age;

alert(ageElems[0]); // [object HTMLInputElement]
</script>
```

<fieldset> как «подформа»

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

```
<body>
  <form id="form">
    <fieldset name="userFields">
      <legend>info</legend>
      <input name="login" type="text">
    </fieldset>
  </form>

  <script>
    alert(form.elements.login); // <input name="login">

    let fieldset = form.elements.userFields;
    alert(fieldset); // HTMLFieldSetElement

    // мы можем достать элемент по имени как из формы, так и из fieldset с ним
    alert(fieldset.elements.login == form.elements.login); // true
  </script>
</body>
```

Сокращённая форма записи: form.name

Есть более короткая запись: мы можем получить доступ к элементу через form[index/name].

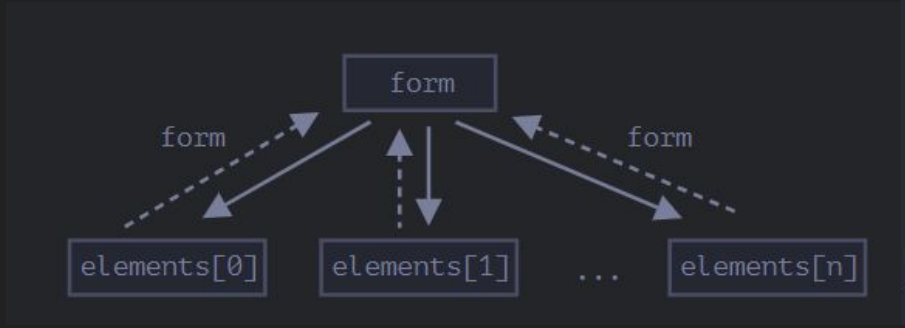
Другими словами, вместо form.elements.login мы можем написать form.login.

Это также работает, но есть небольшая проблема: если мы получаем элемент, а затем меняем его свойство name, то он всё ещё будет доступен под старым именем (также, как и под новым).

```
<form id="form">
  <input name="login">
</form>

<script>
  alert(form.elements.login == form.login); // true, ведь это одинаковые <input>

  form.login.name = "username"; // изменяем свойство name у элемента input

  // form.elements обновили свои имена:
  alert(form.elements.login); // undefined
  alert(form.elements.username); // input

  // а в form мы можем использовать оба имени: новое и старое
  alert(form.username == form.login); // true
</script>
```

Обратная ссылка: element.form

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




```
<form id="form">
  <input type="text" name="login">
</form>

<script>
  // form -> element
  let login = form.login;

  // element -> form
  alert(login.form); // HTMLFormElement
</script>
```

Элементы формы: input и textarea

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

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.

```
input.value = "Новое значение";
textarea.value = "Новый текст";

input.checked = true; // для чекбоксов и переключателей
```

Элементы формы: 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>.

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>.

```
<select id="select">
  <option value="apple">Яблоко</option>
  <option value="pear">Груша</option>
  <option value="banana">Банан</option>
</select>

<script>
  // все три строки делают одно и то же
  select.options[2].selected = true;
  select.selectedIndex = 2;
  select.value = 'banana';
</script>
```

```
<select id="select" multiple>
  <option value="blues" selected>Блюз</option>
  <option value="rock" selected>Рок</option>
  <option value="classic">Классика</option>
</select>

<script>
  // получаем все выбранные значения из select с multiple
  let selected = Array.from(select.options)
    .filter(option => option.selected)
    .map(option => option.value);

  alert(selected); // blues,rock
</script>
```

Элементы формы: new Option

Элемент <option> редко используется сам по себе, но и здесь есть кое-что интересное.

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. Впрочем, обычно ставят оба этих значения в true или не ставят вовсе (т.е. false).

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).

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

```
let option = new Option("Текст", "value");
// создаст <option value="value">Текст</option>
```

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

Фокусировка: focus/blur

- События focus/blur
- Методы focus/blur
- Включаем фокусировку на любом элементе: tabindex
- События focusin/focusout

Элемент получает фокус, когда пользователь кликает по нему или использует клавишу Tab. Также существует HTML-атрибут autofocus, который устанавливает фокус на элемент, когда страница загружается. Есть и другие способы получения фокуса, о них – далее.

Фокусировка обычно означает: «приготовься к вводу данных на этом элементе», это хороший момент, чтобы инициализировать или загрузить что-нибудь.

Момент потери фокуса («blur») может быть важнее. Это момент, когда пользователь кликает куда-то ещё или нажимает Tab, чтобы переключиться на следующее поле формы. Есть другие причины потери фокуса, о них – далее.

Потеря фокуса обычно означает «данные введены», и мы можем выполнить проверку введённых данных или даже отправить эти данные на сервер и так далее.

События focus/blur

Событие focus вызывается в момент фокусировки, а blur – когда элемент теряет фокус.

Используем их для валидации(проверки) введенных данных.

В примере:

Обработчик blur проверяет, введён ли email, и если нет – показывает ошибку.

Обработчик focus скрывает это сообщение об ошибке (в момент потери фокуса проверка повторится).

Современный HTML позволяет делать валидацию с помощью атрибутов required, pattern и т.д. Иногда – это всё, что нам нужно. JavaScript можно использовать, когда мы хотим больше гибкости. А ещё мы могли бы отправлять изменённое значение на сервер, если оно правильное.

```
<style>
  .invalid { border-color: red; }
  #error { color: red }
</style>

Ваш email: <input type="email" id="input">

<div id="error"></div>

<script>
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 = "";
  }
};
</script>
```

Методы focus/blur

Методы elem.focus() и elem.blur() устанавливают/снимают фокус.

Например, запретим посетителю переключаться с поля ввода, если введенное значение не прошло валидацию:

Это сработает во всех браузерах, кроме Firefox (bug).

Если мы что-нибудь введем и нажмём Tab или кликнем в другое место, тогда onblur вернёт фокус обратно.

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

Однако на практике следует хорошо подумать, прежде чем внедрять что-то подобное, потому что мы обычно должны показывать ошибки пользователю, но они не должны мешать пользователю при заполнении нашей формы. Ведь, вполне возможно, что он захочет сначала заполнить другие поля.

```
<style>
  .error {
    background: red;
  }
</style>

Ваш email: <input type="email" id="input">
<input type="text" style="width:280px" placeholder="введите неверный email и кликните сюда">

<script>
  input.onblur = function() {
    if (!this.value.includes('@')) { // не email
      // показать ошибку
      this.classList.add("error");
      // ...и вернуть фокус обратно
      input.focus();
    } else {
      this.classList.remove("error");
    }
  };
</script>
```

Потеря фокуса, вызванная JavaScript

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).
- Если элемент удалить из DOM, фокус также будет потерян. Если элемент добавить обратно, то фокус не вернётся.

Из-за этих особенностей обработчики focus/blur могут сработать тогда, когда это не требуется.

Используя эти события, нужно быть осторожным. Если мы хотим отследить потерю фокуса, которую инициировал пользователь, тогда нам следует избегать её самим.

Включаем фокусировку на любом элементе: tabindex

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.

Пример:

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.

Свойство elem.tabIndex тоже работает

Мы можем добавить tabindex из JavaScript, используя свойство elem.tabIndex. Это даст тот же эффект.

```
Кликните первый пункт в списке и нажмите Tab.
Продолжайте следить за порядком.
Обратите внимание, что много последовательных нажатий
Tab могут вывести фокус из iframe с примером.
<ul>
  <li tabindex="1">Один</li>
  <li tabindex="0">Ноль</li>
  <li tabindex="2">Два</li>
  <li tabindex="-1">Минус один</li>
</ul>

<style>
  li { cursor: pointer; }
  :focus { outline: 1px dashed green; }
</style>
```

События focusin/focusout

События focus и blur не всплывают.

Например, мы не можем использовать onfocus на <form>, чтобы подсветить её:

Пример не работает, потому что когда пользователь перемещает фокус на <input>, событие focus срабатывает только на этом элементе. Это событие не всплывает. Следовательно, form.onfocus никогда не срабатывает.

У этой проблемы два решения.

```
<!-- добавить класс при фокусировке на форме -->
<form onfocus="this.className='focused'">
  <input type="text" name="name" value="Имя">
  <input type="text" name="surname" value="Фамилия">
</form>

<style> .focused { outline: 1px solid red; } </style>
```

Первое решение

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

```
<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>.

```
<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>
```

focus и blur ИТОГО

События focus и blur срабатывают на фокусировке/потере фокуса элемента.

Их особенности:

- Они не всплывают. Но можно использовать фазу перехвата или focusin/focusout.
- Большинство элементов не поддерживают фокусировку по умолчанию. Используйте tabindex, чтобы сделать фокусируемым любой элемент.

Текущий элемент с фокусом можно получить из document.activeElement.

События: change, input, cut, copy, paste

- Событие: change
- Событие: input
- События: cut, copy, paste

Событие: change

Событие change срабатывает по окончании изменения элемента.

Для текстовых <input> это означает, что событие происходит при потере фокуса.

Пока мы печатаем в текстовом поле в примере ниже, событие не происходит. Но когда мы перемещаем фокус в другое место, например, нажимая на кнопку, то произойдет событие change.

Для других элементов: select, input type=checkbox/radio событие запускается сразу после изменения значения.

```
<input type="text" onchange="alert(this.value)">
<input type="button" value="Button">
```

```
<select onchange="alert(this.value)">
  <option value="">Выберите что-нибудь</option>
  <option value="1">Вариант 1</option>
  <option value="2">Вариант 2</option>
  <option value="3">Вариант 3</option>
</select>
```

Событие: input

```
<input type="text" id="input"> oninput: <span id="result"></span>
<script>
  input.oninput = function() {
    result.innerHTML = input.value;
  };
</script>
```

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.

С другой стороны, 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.

Нельзя ничего предотвратить в oninput

Событие input происходит после изменения значения.

Поэтому мы не можем использовать event.preventDefault() там – будет уже слишком поздно, никакого эффекта не будет.

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 копируется/не вставится.

Технически, мы можем скопировать/вставить всё. Например, мы можем скопировать файл из файловой системы и вставить его.

Существует список методов в спецификации для работы с различными типами данных, чтения/записи в буфер обмена.

Но обратите внимание, что буфер обмена работает глобально, на уровне ОС. Большинство браузеров в целях безопасности разрешают доступ на чтение/запись в буфер обмена только в рамках определенных действий пользователя, к примеру, в обработчиках событий onclick.

Также запрещается генерировать «пользовательские» события буфера обмена при помощи dispatchEvent во всех браузерах, кроме Firefox.

```
<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>
```

Отправка формы: событие и метод submit

- Событие: submit
- Метод: submit

При отправке формы срабатывает событие submit, оно обычно используется для проверки (валидации) формы перед ее 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.

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 генерируют 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.

```
<form onsubmit="alert('submit!');return false">
  Первый пример: нажмите Enter: <input type="text" value="Текст"><br>
  Второй пример: нажмите на кнопку "Отправить": <input type="submit" value="Отправить">
</form>
```

Взаимосвязь между submit и click

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.

```
<form onsubmit="alert('submit!');return false">
 <input type="text" size="30" value="Установите фокус здесь и нажмите Enter">
 <input type="submit" value="Отправить" onclick="alert('click')">
</form>
```

Метод: submit

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.

```
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();
```

Шаблон pattern, атрибут required

required: Этот атрибут добавляется к <input>, чтобы указать, что поле должно быть заполнено перед отправкой формы. Если поле остается пустым, браузер не позволит отправить форму и выведет сообщение об ошибке.

pattern: Этот атрибут позволяет задать регулярное выражение, которому должно соответствовать значение поля. Если значение не соответствует заданному шаблону, браузер также выведет сообщение об ошибке.

Используем checkValidity() для проверки встроенной валидации браузера.

Регулярные выражения

Регулярные выражения (regex) — это мощный инструмент для работы с текстовыми данными в JavaScript и других языках программирования. Они используются для поиска, замены и проверки строк на соответствие определенным шаблонам.

Синтаксис: Регулярные выражения в JavaScript создаются с помощью объекта RegExp или с помощью литерала /pattern/

```
// Литерал
const regex = /pattern/;

// С помощью конструктора RegExp
const regex = new RegExp('pattern');
```

Ресурсы

Свойства и методы формы - ТЫК

Фокусировка: focus/blur - ТЫК

События: change, input, cut, copy, paste - ТЫК

Отправка формы: событие и метод submit - ТЫК

Регулярные выражения - ТЫК