Введение в браузерные события

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

Вот список самых часто используемых DOM-событий, пока просто для ознакомления:

События мыши:

- `click` – происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании).
- `contextmenu` – происходит, когда кликнули на элемент правой кнопкой мыши.
- `mouseover / mouseout` – когда мышь наводится на / покидает элемент.
- `mousedown / mouseup` – когда нажали / отжали кнопку мыши на элементе.
- `mousemove` – при движении мыши.

События на элементах управления:

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

Клавиатурные события:

- `keydown` и `keyup` – когда пользователь нажимает / отпускает клавишу.

События документа:

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

CSS events:

- `transitionend` – когда CSS-анимация завершена.

Существует множество других событий. Мы подробно разберём их в последующих главах.

Обработчики событий

Событию можно назначить *обработчик*, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик. Сейчас мы их рассмотрим, начиная с самого простого.

Использование атрибута HTML

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется `on<событие>`.

Например, чтобы назначить обработчик события `click` на элементе `input`, можно использовать атрибут `onclick`, вот так:

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

При клике мышкой на кнопке выполнится код, указанный в атрибуте `onclick`.

Поделиться

Редактировать на GitHub

Обратите внимание, для содержимого атрибута `onclick` используются одинарные кавычки, так как сам атрибут находится в двойных. Если мы забудем об этом и поставим двойные кавычки внутри атрибута, вот так: `onclick="alert("Click!")"`, код не будет работать.

Атрибут HTML-тега – не самое удобное место для написания большого количества кода, поэтому лучше создать отдельную JavaScript-функцию и вызвать её там.

Следующий пример по клику запускает функцию `countRabbits()`:

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Кролик номер " + i);
    }
  }
</script>

<input type="button" onclick="countRabbits()" value="Сч
```

Считать кроликов!

Как мы помним, атрибут HTML-тега не чувствителен к регистру, поэтому `ONCLICK` будет работать так же, как `onClick` и `onCLICK` … Но, как правило, атрибуты пишут в нижнем регистре: `onclick`.

Использование свойства DOM-объекта

Можно назначать обработчик, используя свойство DOM-элемента `on<событие>`.

К примеру, `elem.onclick`:

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

Нажми меня!

Если обработчик задан через атрибут, то браузер читает HTML-разметку, создаёт новую функцию из содержимого атрибута и записывает в свойство.

Этот способ, по сути, аналогичен предыдущему.

Обработчик всегда хранится в свойстве DOM-объекта, а атрибут – лишь один из способов его инициализации.

Эти два примера кода работают одинаково:

1. Только HTML:

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

Кнопка

2. HTML + JS:

```
<input type="button" id="button" value="Кнопка">
<script>
  button.onclick = function() {
    alert('Клик!');
  };
</script>
```

Кнопка

Так как у элемента DOM может быть только одно свойство с именем `onclick`, то назначить более одного обработчика так нельзя.

В примере ниже назначение через JavaScript перезапишет обработчик из атрибута:

```
<input type="button" id="elem" onclick="alert('Было')">
<script>
  elem.onclick = function() { // перезапишет существующ
    alert('Станет'); // выведется только это
  };
</script>
```

Нажми меня

Кстати, обработчиком можно назначить и уже существующую функцию:

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

elem.onclick = sayThanks;
```

Убрать обработчик можно назначением `elem.onclick = null`.

Доступ к элементу через this

Внутри обработчика события `this` ссылается на текущий элемент, то есть на тот, на котором, как говорят, «висит» (т.е. назначен) обработчик.

В коде ниже `button` выводит своё содержимое, используя `this.innerHTML`:

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

Нажми меня

Частые ошибки

Если вы только начинаете работать с событиями, обратите внимание на следующие моменты.

Функция должна быть присвоена как `sayThanks`, а не `sayThanks()`.

```
// правильно
button.onclick = sayThanks;

// неправильно
button.onclick = sayThanks();
```

Если добавить скобки, то `sayThanks()` – это уже вызов функции, результат которого (равный `undefined`, так как функция ничего не возвращает) будет присвоен `onclick`. Так что это не будет работать.

...А вот в разметке, в отличие от свойства, скобки нужны:

```
<input type="button" id="button" onclick="sayThanks()">
```

Это различие просто объяснить. При создании обработчика браузером из атрибута, он автоматически создаёт функцию с *телом из значения атрибута*: `sayThanks()`.

Так что разметка генерирует такое свойство:

```
button.onclick = function() {
  sayThanks(); // содержимое атрибута
};
```

Используйте именно функции, а не строки.

Назначение обработчика строкой `elem.onclick = "alert(1)"` также сработает. Это сделано из соображений совместимости, но делать так не рекомендуется.

Не используйте `setAttribute` для обработчиков.

Такой вызов работать не будет:

```
// при нажатии на body будут ошибки,
// атрибуты всегда строки, и функция станет строкой
document.body.setAttribute('onclick', function() { aler
```

Регистр DOM-свойства имеет значение.

Используйте `elem.onclick`, а не `elem.ONCLICK`, потому что DOM-свойства чувствительны к регистру.

addEventListener

Фундаментальный недостаток описанных выше способов назначения обработчика –- невозможность повесить несколько обработчиков на одно событие.

Например, одна часть кода хочет при клике на кнопку делать её подсвеченной, а другая – выдавать сообщение.

Мы хотим назначить два обработчика для этого. Но новое DOM-свойство перезапишет предыдущее:

```
input.onclick = function() { alert(1); }
// ...
input.onclick = function() { alert(2); } // заменит пре
```

Разработчики стандартов достаточно давно это поняли и предложили альтернативный способ назначения обработчиков при помощи специальных методов `addEventListener` и `removeEventListener`. Они свободны от указанного недостатка.

Синтаксис добавления обработчика:

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

`event`

Имя события, например `"click"`.

`handler`

Ссылка на функцию-обработчик.

`options`

Дополнительный объект со свойствами:

- `once`: если `true`, тогда обработчик будет автоматически удалён после выполнения.
- `capture`: фаза, на которой должен сработать обработчик, подробнее об этом будет рассказано в главе Всплытие и погружение. Так исторически сложилось, что `options` может быть `false/true`, это то же самое, что `{capture: false/true}`.

- `passive`: если `true`, то указывает, что обработчик никогда не вызовет `preventDefault()`, подробнее об этом будет рассказано в главе Действия браузера по умолчанию.

Для удаления обработчика следует использовать `removeEventListener`:

```
element.removeEventListener(event, handler[, options]);
```

Удаление требует именно ту же функцию

Для удаления нужно передать именно ту функцию-обработчик которая была назначена.

Вот так не сработает:

```
elem.addEventListener( "click" , () => alert('Спас
// ....
elem.removeEventListener( "click", () => alert('Сп
```

Обработчик не будет удалён, т.к. в `removeEventListener` передана не та же функция, а другая, с одинаковым кодом, но это не важно.

Вот так правильно:

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

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

Обратим внимание – если функцию обработчик не сохранить где-либо, мы не сможем её удалить. Нет метода, который позволяет получить из элемента обработчики событий, назначенные через `addEventListener`.

Метод `addEventListener` позволяет добавлять несколько обработчиков на одно событие одного элемента, например:

```
<input id="elem" type="button" value="Нажми меня"/>

<script>
  function handler1() {
    alert('Спасибо!');
  };

  function handler2() {
    alert('Спасибо ещё раз!');
  }

  elem.onclick = () => alert("Привет");
  elem.addEventListener("click", handler1); // Спасибо!
  elem.addEventListener("click", handler2); // Спасибо
</script>
```

Как видно из примера выше, можно одновременно назначать обработчики и через DOM-свойство и через `addEventListener`. Однако, во избежание путаницы, рекомендуется выбрать один способ.

Поделиться

Редактировать на GitHub

Обработчики некоторых событий можно назначать только через `addEventListener`

Существуют события, которые нельзя назначить через DOM-свойство, но можно через `addEventListener`.

Например, таково событие `DOMContentLoaded`, которое срабатывает, когда завершена загрузка и построение DOM документа.

```
document.onDOMContentLoaded = function() {
  alert("DOM построен"); // не будет работать
};
```

```
document.addEventListener("DOMContentLoaded", func
  alert("DOM построен"); // а вот так сработает
});
```

Так что `addEventListener` более универсален. Хотя заметим, что таких событий меньшинство, это скорее исключение, чем правило.

Объект события

Чтобы хорошо обработать событие, могут понадобиться детали того, что произошло. Не просто «клик» или «нажатие клавиши», а также – какие координаты указателя мыши, какая клавиша нажата и так далее.

Когда происходит событие, браузер создаёт *объект события*, записывает в него детали и передаёт его в качестве аргумента функции-обработчику.

Пример ниже демонстрирует получение координат мыши из 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("Координаты: " + event.clientX + ":" + event.
  };
</script>
```

Некоторые свойства объекта `event`:

`event.type`

Тип события, в данном случае `"click"`.

`event.currentTarget`

Элемент, на котором сработал обработчик. Значение – обычно такое же, как и у `this`, но если обработчик является функцией-стрелкой или при помощи `bind` привязан другой объект в качестве `this`, то мы можем получить элемент из `event.currentTarget`.

`event.clientX / event.clientY`

Координаты курсора в момент клика относительно окна, для событий мыши.

Есть также и ряд других свойств, в зависимости от типа событий, которые мы разберём в дальнейших главах.

Поделиться

Редактировать на GitHub

Объект события доступен и в HTML

При назначении обработчика в HTML, тоже можно использовать объект `event`, вот так:

```
<input type="button" onclick="alert(event.type)" \
```

Это возможно потому, что когда браузер из атрибута создаёт функцию-обработчик, то она выглядит так: `function(event) { alert(event.type) }`. То есть, её первый аргумент называется `"event"`, а тело взято из атрибута.

Объект-обработчик: handleEvent

Мы можем назначить обработчиком не только функцию, но и объект при помощи `addEventListener`. В этом случае, когда происходит событие, вызывается метод объекта `handleEvent`.

К примеру:

```
<button id="elem">Нажми меня</button>

<script>
  elem.addEventListener('click', {
    handleEvent(event) {
      alert(event.type + " на " + event.currentTarget);
    }
  });
</script>
```

Как видим, если `addEventListener` получает объект в качестве обработчика, он вызывает `object.handleEvent(event)`, когда происходит событие.

Мы также можем использовать класс для этого:

```
<button id="elem">Нажми меня</button>

<script>
  class Menu {
    handleEvent(event) {
      switch(event.type) {
        case 'mousedown':
          elem.innerHTML = "Нажата кнопка мыши";
          break;
        case 'mouseup':
          elem.innerHTML += "...и отжата.";
          break;
      }
    }
  }

  let menu = new Menu();
  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);
</script>
```

Здесь один и тот же объект обрабатывает оба события. Обратите внимание, мы должны явно назначить оба обработчика через `addEventListener`. Тогда объект `menu` будет получать события `mousedown` и `mouseup`, но не другие (не назначенные) типы событий.

Поделиться

Редактировать на GitHub

Метод `handleEvent` не обязательно должен выполнять всю работу сам. Он может вызывать другие методы, которые заточены под обработку конкретных типов событий, вот так:

```
<button id="elem">Нажми меня</button>

<script>
  class Menu {
    handleEvent(event) {
      // mousedown -> onMousedown
      let method = 'on' + event.type[0].toUpperCase() +
      this[method](event);
    }

    onMousedown() {
      elem.innerHTML = "Кнопка мыши нажата";
    }

    onMouseup() {
      elem.innerHTML += "...и отжата.";
    }
  }

  let menu = new Menu();
  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);
</script>
```

Теперь обработка событий разделена по методам, что упрощает поддержку кода.

Итого

Есть три способа назначения обработчиков событий:

1. Атрибут HTML: `onclick="..."`.
2. DOM-свойство: `elem.onclick = function`.
3. Специальные методы: `elem.addEventListener(event, handler[, phase])` для добавления, `removeEventListener` для удаления.

HTML-атрибуты используются редко потому, что JavaScript в HTML-теге выглядит немного странно. К тому же много кода там не напишешь.

DOM-свойства вполне можно использовать, но мы не можем назначить больше одного обработчика на один тип события. Во многих случаях с этим ограничением можно мириться.

Последний способ самый гибкий, однако нужно писать больше всего кода. Есть несколько типов событий, которые работают только через него, к примеру `transitionend` и `DOMContentLoaded`. Также `addEventListener` поддерживает объекты в качестве обработчиков событий. В этом случае вызывается метод объекта `handleEvent`.

Не важно, как вы назначаете обработчик – он получает объект события первым аргументом. Этот объект содержит подробности о том, что произошло.

Мы изучим больше о событиях и их типах в следующих главах.

Задачи

Скрыть элемент по нажатию кнопки

важность: 5

Добавьте JavaScript к кнопке `button`, чтобы при нажатии элемент `<div id="text">` исчезал.

Демо:

Нажмите, чтобы спрятать текст
Текст

Открыть песочницу для задачи.

решение

Спрятать себя

важность: 5

Создайте кнопку, которая будет скрывать себя по нажатию.

Например: Нажмите, чтобы спрятать

решение

Какой обработчик запустится?

важность: 5

В переменной `button` находится кнопка. Изначально на ней нет обработчиков.

Который из обработчиков запустится? Что будет выведено при клике после выполнения кода?

```
1  button.addEventListener("click", () => alert("1"));
2
3  button.removeEventListener("click", () => alert("1"));
4
5  button.onclick = () => alert(2);
```

решение

Передвиньте мяч по полю

важность: 5

Пусть мяч перемещается при клике на поле, туда, куда был клик, вот так:

Нажмите на поле для перемещения мяча.

Требования:

- Центр мяча должен совпадать с местом нажатия мыши (если это возможно без пересечения краёв поля);
- CSS-анимация желательна, но не обязательна;
- Мяч ни в коем случае не должен пересекать границы поля;
- При прокрутке страницы ничего не должно ломаться;

Заметки:

- Код должен уметь работать с различными размерами мяча и поля, не привязываться к каким-либо фиксированным значениям.

- Используйте свойства `event.clientX/event.clientY` для определения координат мыши при клике.

Открыть песочницу для задачи.

решение

Создать раскрывающееся меню

важность: 5

Создать меню, которое по нажатию открывается либо закрывается:

▸ Сладости (нажми меня)!

P.S. HTML/CSS исходного документа можно и нужно менять.

Открыть песочницу для задачи.

решение

Добавить кнопку закрытия

важность: 5

Есть список сообщений.

При помощи JavaScript для каждого сообщения добавьте в верхний правый угол кнопку закрытия.

Результат должен выглядеть, как показано здесь:

Лошадь [x]

Домашняя лошадь — животное семейства непарнокопытных, одомашненный и единственный сохранившийся подвид дикой лошади, вымершей в дикой природе, за исключением небольшой популяции лошади Пржевальского.

Осёл [x]

Домашний осёл (лат. Equus asinus asinus), или ишак, — одомашненный подвид дикого осла (Equus asinus), сыгравший важную историческую роль в развитии хозяйства и культуры человека и по-прежнему широко в хозяйстве многих развивающихся стран.

Кошка [x]

Кошка, или домашняя кошка (лат. Felis silvestris catus), — домашнее животное, одно из наиболее популярных(наряду с собакой) «животных-компаньонов». Являясь одиночным охотником на грызунов и других мелких животных, кошка —

Открыть песочницу для задачи.

решение

Карусель

важность: 4

Создайте «Карусель» –- ленту изображений, которую можно листать влево-вправо нажатием на стрелочки.

В дальнейшем к ней можно будет добавить анимацию, динамическую подгрузку и другие возможности.

P.S. В этой задаче разработка структуры HTML/CSS составляет 90% решения.

Открыть песочницу для задачи.

решение

Проводим курсы по JavaScript и фреймворкам.

Комментарии

перед тем как писать...

© 2007—2020 Илья Кантор | о проекте | связаться с нами | пользовательское соглашение | политика конфи