Материалы занятия

Курс: Разработка интерфейса на JavaScript

Дисциплина: Основы JavaScript

Тема занятия №19: Практика по обработке событий

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:

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

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

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="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>
```

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>
```

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')" value="Нажми меня">
<script>
  elem.onclick = function() { // перезапишет существующий обработчик
    alert('Станет'); // выведется только это
  };
</script>
```

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</button>
```

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() { alert(1) });
```

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 предыдущий обработчик
```

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

Для удаления обработчика следует использовать removeEventListener:

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

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.clientY);
  };
</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.

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)" value="Тип события">
```

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.

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.

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