Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебная дисциплина «Программные средства создания Internet-приложений»

Инструкция
по выполнению лабораторной работы
«Использование разных моделей событий. Остановка всплытия и перехват событий»

Минск
2021

Лабораторная работа № 24

Тема работы: Использование разных моделей событий. Остановка всплытия и перехват событий

1. Цель работы

Формирование умений назначения обработчиков событий в сценариях JavaScript, использования разных моделей событий.

2. Задание

Выполнить задания в соответствии с порядком выполнения лабораторной работы.

3. Оснащение работы

ПК, редактор исходного кода, браузер.

4. Основные теоретические сведения

Событие в JavaScript – это определённое действие, которое вызвано либо пользователем, либо браузером. К ним относятся движение мыши, нажатия на кнопки мыши и клавиатуры, наведение фокуса на элемент, изменение значения в каком-нибудь текстовом поле, изменение размеров окна браузера и так далее.

Для реакции на определённые события (действия пользователя) назначаются обработчики событий.

Обработчик события – это функция, которая выполняется в тот момент, когда происходит событие, и определяет, что будет происходить при возникновении определённого события. Обработчики событий имеют следующий общий вид:

```
onНазваниеСобытия
```

Обработчики событий могут быть привязаны к объекту Element, Document, Window и т.д. Затем, в тот момент, когда происходит какое-либо событие, создаётся объект Event (событие), который передаётся в качестве аргумента обработчику события.

Назначить обработчик события можно несколькими способами.

1. Самый простой способ задания обработчика – это использование атрибутов событий прямо в HTML-коде.

Значением атрибутов событий указывается код JavaScript или название готовой функции. Если указывается просто код JavaScript, то браузер автоматически создаёт анонимную функцию, телом которой является исходный код.

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

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

```
document.getElementById('bt').onclick;   //ƒ onclick(event) {alert('Клик!')}
document.getElementById('bt').getAttribute('onclick');     //"alert('Клик!')"
document.getElementById('bt').onclick=function (){alert('Новый клик!')};     //ƒ ()
                                                          //{alert('Новый клик!')}
document.getElementById('bt').setAttribute(`onclick`,`alert('Супер клик!')`);      //
                                                                    //Супер клик!
```

2. Ещё один способ установки обработчика события – это прямое использование DOM свойств, то есть сразу записывать обработчик в свойство элемента.

Значением свойства обязательно должна быть функция.

```
let par = document.body.firstElementChild;
par.onmouseover = function () {
this.style.background = 'green';
};
```

Использование HTML-атрибутов или DOM-свойств позволяет устанавливать только по одному обработчику на каждое событие.

3. Для установки и удаления любого количества обработчиков на одно событие используются методы **addEventListener()** и **removeEventListener()**.

Метод **addEventListener()** добавляет обработчик события.

```
element.addEventListener(имя_события, обработчик[,options])
```

имя_события – название события в кавычках;

обработчик – функция-обработчик события (можно указать анонимную функцию или ссылку на готовую);

options – дополнительный объект со свойствами:

once: если true, тогда обработчик будет автоматически удалён после выполнения;

capture: фаза, на которой должен сработать обработчик: (true – на стадии перехвата, false (используется по умолчанию) – на стадии всплытия). Options может быть false/true, это то же самое, что {capture: false/true};

passive: если true, то указывает, что обработчик никогда не вызовет preventDefault().

```
<body>
<input type="button" id='button' value="Кнопка">
</body>

<script>
let button = document.body.firstElementChild;
button.addEventListener('click', function() {alert('Обработчик 1');});
button.addEventListener('click', function() {alert('Обработчик 2');});
</script>
```

Метод **removeEventListener()** используется для удаления обработчика события, назначенного с помощью метода **addEventListener()**. Обязательно должны быть указаны те значения, которые использовались при назначении обработчика.

```
element.removeEventListener(имя_события, обработчик[,options])
```

имя_события – название события в кавычках;

обработчик – функция-обработчик события. Должна указываться именно ссылка, чтобы браузер понимал, какой обработчик нужно удалить. Соответственно, если обработчик был назначен напрямую (анонимной функцией), то его удалить не удастся;

options – необязательный аргумент. Указывается, если использовался при назначении обработчика.

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

```
<body>
<input type="button" value="Кнопка">
</body>
<script>
function handler() {alert('Обработчик 1');}
let button = document.body.firstElementChild;
/* добавление обработчиков */
```

```
button.addEventListener('click', handler);
button.addEventListener('click', function() {alert('Обработчик 2');});
/* удаление обработчиков */
button.removeEventListener('click', handler);
button.removeEventListener('click', function() {alert('Обработчик 2');});
</script>
```

Второй обработчик остался, так как он назначен анонимной функцией, т.е. т.к. в removeEventListener передана не та же функция, а другая, с одинаковым кодом.

Если функцию обработчик не сохранить где-либо, возможности её удалить не будет. Нет метода, который позволяет получить из элемента обработчики событий, назначенные через addEventListener.

Метод addEventListener позволяет добавлять несколько обработчиков на одно событие одного элемента.

```
<input type="button" id='bt' value="Кнопка">
<script>
function funk1() {
    console.log('Добрый день!!');
  };

  function funk2() {
    console.log ('Добрый вечер!');
  }
  bt.onclick = () => console.log ("Доброе утро");
  bt.addEventListener("click", funk1);
  bt.addEventListener("click", funk2);
</script>
```

Можно одновременно назначать обработчики и через DOM-свойство и через addEventListener.

Следующий пример показывает, как объект Event передаётся обработчику события и может быть использован внутри него.

```
window.addEventListener("keydown", foo, false);

function foo(event) {
  // параметр event неявно инициализируется объектом Event
  alert(event);
}
```

Само исходное событие для всех обработчиков является общим. Его обработка делится на три стадии:

- стадия перехвата (capturing stage);
- стадия цели (target stage);
- стадия всплытия (bubbling stage).

Элемент, на котором возникает событие, называется **целевым**.

При возникновении события браузер поочерёдно проходит от верхнего элемента DOM-дерева (document) вниз через все промежуточные элементы к целевому. Эта стадия обработки называется стадией **перехвата**. Обработчики запускаются до того, как событие дойдёт до целевого элемента. Событие как бы перехватывается. Отсюда и следует такое название стадии.

Обработчики события выполняются только в том случае, если для них задано выполнение на стадии перехвата. Для этого необходимо использовать значение **true** для третьего атрибута метода **addEventListener()**. Это единственный способ использовать обработчик на стадии перехвата.

После того, как событие опустилось до целевого элемента, стадия перехвата завершается и выполняются обработчики целевого элемента. Это вторая стадия – стадия **цели**. Очерёдность выполнения обработчиков на целевом элементе зависит только от очерёдности их назначения.

Далее начинается последняя стадия. Событие проходит поочерёдно от целевого элемента через цепочку родителей до корневого элемента документа. Событие продвигается вверх по DOM-дереву. Оно будто всплывает. Отсюда и название – стадия **всплытия**.

На данной стадии выполняются все остальные обработчики. Обработчики, назначенные с помощью атрибутов событий и DOM-свойств, всегда выполняются на стадии всплытия. В методе addEventListener() для использования стадии всплытия можно просто опустить третий аргумент.

Интерфейс события объектной модели документа (DOM) доступен только через объект Event, который передаётся в качестве аргумента в обработчик события.

Новый объект события встроенного класса Event можно создать с помощью конструктора **Event()**:

```
event = new Event(type[, options]);
```

type - строковое значение типа события, например, "click" или же любой придуманный – "my-event", "look" и.т.д.;

options (необязательный) - объект с необязательными свойствами:

"bubbles": логическое значение (Boolean) указывающее – будет ли событие всплывающим. По умолчанию false;

"cancelable": логическое значение (Boolean) указывает, может ли событие быть отменено. По умолчанию false;

"composed": логическое значение Boolean указывающее – будет ли событие всплывать наружу за пределы shadow root. По умолчанию false.

```
let evt = new Event("look", {"bubbles":true, "cancelable":false});

создано всплывающее событие, которое не может быть отменено
```

После того, как объект события создан, необходимо запустить его на элементе, вызвав метод elem.dispatchEvent(event):

```
document.dispatchEvent(evt);

// событие может быть инициализировано на любом элементе, а не только документе
myDiv.dispatchEvent(evt);
```

Затем обработчики отреагируют на него, как будто это обычное браузерное событие.

Для собственных событий необходимо использовать addEventListener , т.к. on<event>-свойства существуют только для встроенных событий, то есть elem.onlook не сработает.

```
<button id="elem" onlook="alert('look!');">Автоклик</button> // не сработает
```

Свойства интерфейса Event

Свойство	Описание
Bubbles	Возвращает логическое значение, которое указывает, является ли событие всплывающим `event.bubbles`
Cancelable	Возвращает логическое значение, указывающее, является ли событие отменяемым `event.cancelable`
cancelBubble	Установка логического значения true перед возвратом из обработчика событий прекращает дальнейшую передачу текущего события (предотвращает всплытие по дереву DOM). Свойство является алиасом метода stopPropagation() `event.cancelBubble`

composed	Возвращает логическое значение, которое указывает, будет ли событие распространяться через границу между теневой модели DOM в стандартную модель DOM `event.composed`
currentTarget	Возвращает целевой объект события, обрабатываемого в настоящее время `event.currentTarget`
defaultPrevented	Получает значение, указывающее, следует ли отменить действия по умолчанию. True - действия по умолчанию должны быть отменены, false - действия по умолчанию разрешаются `event.defaultPrevented`
eventPhase	Указывает, какая фаза события, в настоящее время проверяется `event.eventPhase`
Target	Ссылается на элемент, который является целевым объектом данного события `event.target`
timeStamp	Получает время в миллисекундах, когда произошло событие `event.timeStamp`
Type	Возвращает строковое значение, которое содержит тип события `event.type`
isTrusted	Получает значение, указывающее, было ли событие инициировано в браузере (события браузера и пользовательские события) или в сценарии с использованием метода dispatchEvent() `event.isTrusted`

Методы объекта Event

Метод	Описание
composedPath()	Возвращает путь события, представляющий собой массив объектов, на которых будут вызваны обработчики событий
preventDefault()	Отменяет действие события по умолчанию, если оно является отменяемым, без остановки дальнейшего распространения события
stopImmediatePropagation()	Предотвращает любое дальнейшее распространение события
stopPropagation()	Предотвращает дальнейшее распространение текущего события

Если обработчик события установлен с помощью метода **addEventListener()**, тогда для отмены действий браузера по умолчанию необходимо использовать метод **event.preventDefault()**.

Вызов **preventDefault()** во время любой стадии обработки события отменяет действия браузера. Действия назначенных обработчиков данный метод не отменяет.

```
<body>
<input type="text" placeholder="Введите текст">
</body>

<script>
document.body.firstElementChild.onkeypress = function(event) {
event.preventDefault();
alert('Отменено');
}
</script>
```

Действия браузера можно отменить не для всех событий. Вызов **preventDefault()** на **неотменяемом** событии результата не даст.

Для остановки выполнения события на текущем элементе используется метод ***event*.stopPropagation()**. Данный метод отменяет выполнение обработчиков тех элементов, до которых событие не дошло. Однако, все обработчики текущего элемента будут выполнены даже после вызова stopPropagation().

Метод **stopPropagation()** не отменяет действия браузера по умолчанию.

```
<body>
  <form name="test">
    <input type="text" placeholder="Введите текст" onkeypress="alert('input')">
  </form>
</body>

<script>
  document.forms.test.addEventListener('keypress', function(event) {
    alert('form 1');
    event.stopPropagation();
  }, true);
  document.forms.test.addEventListener('keypress', function(event) {
    alert('form 2');
  }, true);
</script>
```

Чтобы отменить выполнение всех невыполненных обработчиков события, даже назначенных текущему элементу, используется метод **event.stopImmediatePropagation()**.

Метод **stopImmediatePropagation()** не отменяет действия браузера по умолчанию.

```
<body>
<form name="test">
<input type="text" placeholder="Введите текст" onkeypress="alert('input')">
</form>
</body>

<script>
document.forms.test.addEventListener('keypress', function(event) {
alert('form 1');
event.stopImmediatePropagation();
}, true);
document.forms.test.addEventListener('keypress', function(event) {
alert('form 2');
}, true);
</script>
```

Помимо выполнения назначенных обработчиков, события могут вызывать действия браузера по умолчанию. Эти действия как бы подразумеваются сами собой в зависимости от элемента страницы.

Событие	Действие браузера
Click	Клик по ссылке вызывает переход на новую страницу.
Click	Клик по неактивному полю для ввода текста делает его активным.
contextmenu	Клик правой кнопкой мыши открывает контекстное меню.
Dblclick	Двойной клик на тексте выделяет его.
mousedown	Нажатие левой кнопки мыши и удержание её над текстом начинает его выделение.
mousewheel	Движение колёсика мыши вызывает прокрутку страницы.
Keydown	Нажатия на кнопки клавиатуры внутри текстового поля приводят к набору текста.

Событие	Действие браузера
Keydown	Нажатие на кнопку **Enter** в активном поле вызывает отправку формы на сервер.

Обычно, если на конкретное событие устанавливается обработчик, то действия браузера по умолчанию не нужны. Тогда их просто можно отменить. Отменить действия браузера по умолчанию можно простым вызовом **return false** в конце обработчика. Но это можно использовать только внутри обработчиков, установленных через HTML-атрибут или DOM-свойство.

```
<body>
<label>Попробуйте поставить галочку <input type="checkbox" onclick="return
false;"></label>
</body>
```

Метод **dispatchEvent()** возвращает значение false, если хотя бы один из выполняемых обработчиков вызывает event.preventDefault() (при создании объекта события должно быть указано свойство cancelable: true). В остальных случаях возвращается true.

```
<html>
<head>
<title>События</title>
</head>
<body>
<input type="button" onclick="event.preventDefault();" value="Кнопка">
</body>
</html>

<script>
var button = document.body.firstElementChild;
var event_1 = new Event('click', {cancelable: true});
alert(button.dispatchEvent(event_1)); /* false, действие отменено */
var event_2 = new Event('click', {cancelable: false});
alert(button.dispatchEvent(event_2)); /* true, действие не отменено */
</script>
```

Всплытие и перехват событий позволяет реализовать один из самых важных приёмов разработки – делегирование.

Слушатели событий выставляются после загрузки страницы. Так что, когда происходит открытие сайта впервые, браузер скачивает, считывает и выполняет JavaScript.

Идея делегирования в том, что если имеется много элементов, события на которых нужно обрабатывать похожим образом, или если элемент добавляется в DOM уже после загрузки страницы, то, вместо того, чтобы назначать обработчик каждому, назначается один обработчик на общего предка.

```
!DOCTYPE html>
<html>
<head>
<title>События</title>
<style>
.one {font-family:2em}
</style>
</head>
<body>
<ul id='ul'>
 <li>Первый пункт</li>
 <li>Второй пункт</li>
 <li>Третий пункт</li>
</ul>
<input type='button' id='but' value='+li' onclick='addLi()'>
<script>
function addLi(){
```

```
let ul=document.getElementsByTagName('ul')[0];
let li=document.createElement('li');
li.innerHTML="Очередной пункт"
ul.append(li);
};
function toggleDone (event) {
   event.target.style.color='red';
     }
function toggleBack (event) {
   event.target.style.color='black';
     }
document.getElementsByTagName('ul')[0].onmouseover=toggleDone;
document.getElementsByTagName('ul')[0].onmouseout=toggleBack;
</script>
</body>
</html>
```

5. Порядок выполнения работы

1. Создать веб-документ, содержащий текстовое поле для ввода. При вводе очередного символа в тестовое поле реализуйте его вывод на веб-страницу. Вывод должен осуществляться без перезагрузки страницы.

2. Добавьте на веб-страницу ссылку. Реализуйте для нее отмену перехода по ссылке по клику. Добавьте для ссылки 5 обработчиков для разных событий таким образом, чтобы при наступлении соответствующего события выводилось сообщение с типом события.

3. Добавьте 5 произвольных HTML-элементов на веб-страницу. Для каждого элемента назначьте не менее двух обработчиков (можно как на один, так и на разные типы событий). При назначении обработчиков событий использовать возможность их выполнения на различных стадиях (всплытия и перехват).

С помощью диалогового окна реализуйте ввод пользователем события, которое необходимо сгенерировать, и элемент, для которого это событие должно быть вызвано.

4*. Создать на веб-странице таблицу. Реализовать добавление новых строк в таблицу по клику на любую строку: новая строка должна добавляться сразу после той строки, на которой произошло событие (например, клик по первой строке встраивает вторую строку, клик по пятой – шестую, т.е. всегда должна добавляться последующая строка).

По двойному клику на ячейке таблицы реализовать плавную смену фона. Предусмотреть вариант, если фон ячейке уже установлен, двойной клик вернет фон в исходное состояние.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

_____

_____

7. Контрольные вопросы и задания

1. Что представляет собой событие?
2. Для каких целей могут быть использованы события на веб-странице?
3. Назовите способы задания обработчиков событий. В чем отличие?
4. Для чего предназначен объект Event, и как его получить?
5. Перечистите свойства объекта Event.
6. Опишите назначение методов объекта Event.

8. Рекомендуемая литература

1. **JAVASCRIPT.RU** [Электронный ресурс] / Современный учебник JavaScript – 2007—2020 Илья Кантор. – Режим доступа: https://learn.javascript.ru. – Дата доступа: 04.03.2020.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. 4-е изд. – СПб.: Питер, 2018.
3. **Симпсон, К.** ES6 и не только / К. Симпсон. – СПб.: Питер, 2017.
4. **Хавербеке, М.** Выразительный JavaScript. Современное веб-программирование / М. Хавербеке – СПб.: Питер, 2019.