Генерация пользовательских событий. Учебник JS

Можно не только назначать обработчики, но и генерировать события из JavaScript-кода. Пользовательские события могут быть использованы при создании графических компонентов. Встроенные классы для событий формируют иерархию аналогично классам для DOM-элементов. Её корнем является встроенный класс **Event**.

Событие встроенного класса Event можно создать так:

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

Где:

- **type** – тип события, строка, например "click" или же любой придуманный нами – "my-event".
- **options** – объект с тремя необязательными свойствами:
  - **bubbles**: true/false – если true, тогда событие всплывает.
  - **cancelable**: true/false – если true, тогда можно отменить действие по умолчанию.
  - **composed**: true/false – если true, тогда событие будет всплывать наружу за пределы Shadow DOM.

  По умолчанию все три свойства установлены в **false**: {bubbles: false, cancelable: false, composed: false}.

После того, как объект события создан, мы должны запустить его на элементе, вызвав метод

```
elem.dispatchEvent(event).
```

Можно легко отличить «настоящее» событие от сгенерированного кодом.
Свойство **event.isTrusted** принимает значение **true** для событий, порождаемых реальными действиями пользователя, и **false** для генерируемых кодом.

Для некоторых конкретных типов событий есть **свои специфические конструкторы**. Вот небольшой список конструкторов для различных событий пользовательского интерфейса, которые можно найти в спецификации UI Event:

- UIEvent
- FocusEvent
- MouseEvent
- KeyboardEvent
- …

Генерация пользовательских событий. Класс Event Emitter - ВебКадеми

Напишем специальный класс **EventEmitter** для создания пользовательских событий.

```
class EventEmitter {
 constructor() {
  this.events = {};
 }

// Метод для запуска события с именем eventName и переданными данными data.
 emit(eventName, data) {
  const event = this.events[eventName];
  if (event) {
   event.forEach((fn) => {
    // Запускаем все функции для события с именем eventName и убираем там контекст()
    // Контекст this в данном случае:  EventEmitter {events: {…}}
    fn.call(null, data);
   });
  }
 }

// Создаем событие с именем eventName и функцией fn
 subscribe(eventName, fn) {
  if (!this.events[eventName]) {
   this.events[eventName] = [];
   }

  // Добавляем  в массив функции из всех подписок на это событие
  this.events[eventName].push(fn);

  // Возвращаем отписку на событие (удалили из events событие, у которого функция - fn)
  return () => {
   this.events[eventName] = this.events[eventName].filter((eventFn) => fn !== eventFn);
  };
 }
```

```
}
```

Применение в коде:

По клику на кнопку вывести в заголовок имя из инпута.

```
let input = document.querySelector('input[type="text"]');
let button = document.querySelector("button");
let h1 = document.querySelector("h1");

let emitter = new EventEmitter();  // Создаем новое событие нашего класса

// Создаем подписку на событие. 1-ый аргумент – имя события, 2-ой – функция, которую нужно выполнить после наступления этого события
const subscribe1 = emitter.subscribe("event:name-changed", (obj) => {
  h1.innerHTML = `Your name is: ${obj.name}`;
});

// Создадим еще одну подписку
const subscribe2 = emitter.subscribe("event:name-changed", () => {
  alert("Second function subscribed");
});

console.log("emitter: ", emitter);
// emitter:
        EventEmitter {events: {…}}
                events:
                        event:name-changed: Array(2)
                                0: (obj) => { h1.innerHTML = `Your name is: ${obj.name}`; }
                                1: () => { alert("Second function subscribed"); }
                                length: 2
                                [[Prototype]]: Array(0)
                        [[Prototype]]: Object
                [[Prototype]]: Object

//subscribe1();  // Запустить, если хотим отменить подписку
//subscribe2();  // Запустить, если хотим отменить подписку

button.addEventListener("click", () => {
    // Запускаем событие с именем  event:name-changed и некоторыми данными – аргументами для функции. Заметим, что запускать мы его можем не только по клику на кнопку, но и в каком-нибудь другом месте, н-р, в консоли.
    emitter.emit("event:name-changed", { name: input.value });
});
```