Урок 4.7 – 4.8. Promise, fetch API, работа с DOM, данные в браузере, setTimeout, setInterval

`setTimeout`

Функция позволяет выполнить какой-либо код с задержкой.

```
setTimeout(() => {
    // код в этой функции выполнится с задержкой
}, 1000);
```

- Задержка пишется в милисекундах - 1000 - это 1 секунда
- Значение 1000 - неточное. Это минимальное время, через которое выполнится эта функция. Реальное время паузы может оказаться дольше

`setInterval`

Функция позволяет выполнять какой-либо код периодически через определённый интервал времени.

```
// Данный код будет выводить в консоль hello world
// через каждые 1000 милисекунд
setInterval(() => {
    console.log('Hello world!');
}, 1000);
```

- Время 1000 милисекунд - это минимальное время интервала. Реальный интервал может оказаться больше.
- Выполнение кода не остановится, пока вы не перезагрузите страницу

- Можно выключить выполнение заданной функции, но для этого внутреннюю функцию нужно задать явно:

```
function handler() {
    console.log('Hello World');
}

setInterval(handler, 1000);

// чтобы очистить, нужно использовать функцию clearInterval,
// указав в качестве параметра ту же функцию, которая была задана
// при создании интервала
clearInterval(hanlder);
```

Promise

- Промисы используются для обработки асинхронных вычислений.
- Чаще всего используются для получения данных с сервера: обычно время ответа точно угадать нельзя, поэтому нужно подождать, пока данные придут и затем выполнить какой-то код
- Но пример промиса можно сделать и без получения данных с сервера:

```
const promise = new Promise((resolve, reject) => {
    // Код внутри Promise выполнится спустя секунду из-за timeout
    // Мы как бы имитируем ответ от сервера с задержкой
   setTimeout(() => {
       // После вызова функции resolve начнётся выполнение функции,
       // помещённой в then (ниже в коде)
       resolve();
   }, 1000);
});

promise.then(() => {
    // выведется после вызова resolve()
    console.log('Промис завершился успешно');
}).catch(() => {
    // выведется, если была вызвана функция reject() вместо resolve()
    console.log('Промис завершился неудачно');
}).finally(() => {
    // Выведется в любом случае после then или catch
    console.log('Промис завершился');
});
```

fetch API

- fetch используется для получения данных с сервера, API и т.п. по URL.
- fetch возвращает Promise, поэтому конструкция выглядит следующим обраазом:

```
fetch('http://example.com/movies.json')
    .then(response => response.json())
    .then((data) => {
        console.log(data);
    });
```

События браузера. Всплытие и погружение. Работа с DOM

Получить объект элемента

По `id`

```
const element = document.getElementById('someId');
```

По селектору

```
// querySelector выбирает первый элемент, совпавший с указанным
// селектором
const elementByClass = document.querySelector('.some-class');
const elementById = document.querySelector('#someId');
const elementByMultipleClasses = document.querySelector('.some-class.some-other-class');
```

Обработчики событий

Добавить обработчик события

```
const element = document.querySelector('.some-element');
element.addEventListener('click', (event) => {
    // функция выполнится при клике на элемент
});
```

Событие ввода в `input`

```
const input = document.querySelector('input[name=phone]');
input.addEventListener('input', (event) => {
   // вызывается каждый раз при изменении
   // значения input
   // event.target.value - текущее значение input
    console.log(event.target.value);

   // Удалили все пробелы и записали как
   // новое значение input
    event.target.value = event.target.value.replaceAll(' ', '');
});
```

Событие нажатия клавиши мыши

```
const element = document.querySelector('.some-element');
element.addEventListener('mousedown', (event) => {
   // вызовется при нажатии кнопки мыши, если курсор на
   // данном элементе.
   // Отрабатывает "на щелчок" мыши
});
```

Событие отпускания клавиши мыши

```
const element = document.querySelector('.some-element');
element.addEventListener('mouseup', (event) => {
   // вызовется при поднятии кнопки мыши, если курсор на
   // данном элементе.
   // Отрабатывает при отщёлкиваии клавиши мыши (ход вверх)
});
```

Смена стилей элемента через JS

```
const element = document.querySelector('.some-element');
// добавили красную границу в 1 px
element.style.border = '1px solid #ff0000';
// Скрыли элемент, добавив display: none
element.style.display = 'none';
// Убрали значение display: теперь оно
// по умолчанию для этого элемента, или то,
// которое задано в классах
element.style.display = '';
```

Значения, добавленные в `element.style`, добавляются как стили, указанные в атрибуте `style` в html:

```
<div style="border: 1px solid #ff0000"></div>
```

Соответственно, эти значения переопределяют стили из классов, потому что стили в атрибуте `style` самые приоритетные.

Управление классами элемента

```
const element = document.querySelector('.some-element');
// Добавили класс some-class к элементу element
// Если класс описан в CSS, то его стили применятся к этому
// элементу
element.classList.add('some-class');
// Добавили класс some-other-class к элементу element
element.classList.add('some-other-class');
// Удалили класс some-other-class
element.classList.remove('some-other-class');
```

Смена значений атрибутов через JS

```
<!-- HTML -->
<input name="email" type="email">
```

```
// JavaScript
const input = document.querySelector('input[name=email]');
input.name = 'phone';
input.type = 'text';
```

Управление событиями

Стандартное (default) поведение событие, например, отправку формы на нажатие кнопки button с типом submit, можно остановить:

```
const button = document.querySelector('.button');
button.addEventListener('click', (event) => {
    event.preventDefault();
});
```

Можно предотвратить дальнейшее всплытие события: на родительских элементах оно уже не выполнится:

```
const element = document.querySelector('.some-element');
// Событие click выполнится для текущего элемента,
// но не выполнится для всех родительских элементов
element.addEventListener('click', (event) => {
    event.stopPropagation();
});
```

LocalStorage, SessionStorage, Cookies

Общее

Все хранилища используются для сохранения пользовательских данных в браузере, чтобы при повторном запуске скрипта они были доступны.

SessionStorage

- Работает только в пределах одной вкладки
- Нужен для сохранения данных при обновлении страницы
- Если вкладку закрыть, все данные, которые сохранили в SessionStorage пропадут

```
// Установить значение value с ключом key в SessionStorage
sessionStorage.setItem('key', 'value');

// Получить значение из SessionStorage с ключом key
const storageValue = sessionStorage.getItem('key');
```

LocalStorage

- В отличие от SessionStorage, сохраняет данные до тех пор, пока пользователь сам их не удалит
- Может использоваться для хранения аналитических данных, а также для предзаполнения форм или списка недавно просмотренных товаров
- Ассоциируется с доменом: данные, сохранённые другим доменом, получить не удастся

```
// Установить значение value с ключом key в LocalStorage
localStorage.setItem('key', 'value');
// Получить значение из LocalStorage с ключом key
const storageValue = localStorage.getItem('key');
```

Cookies

- В отличие от LocalStorage, куки отправляются с запросом на сервер (бэкенд)
- В отличие от LocalStorage, куки могут быть установлены непосредственно сервером (бэкендом) в ответе на запрос.
- Cookies нельзя поставить, если у вас просто открыт файл index.html в браузере, обязательно нужен локальный сервер (адрес должен быть localhost:8000, где 8000 - порт)

```
// Данное выражение ДОБАВЛЯЕТ новую пару ключ-значение в cookies.
// При этом старые значения остаются
document.cookie = 'key' + '=' + 'value';
// Получить cookies
const cookies = document.cookie;
// Возвращаются в формате строки:
// 'key=value; key2=value2; key3=value3'
```

`js-cookie`

- С Cookies не очень удобно работать в чистом JS, поэтому резонно использовать для этого библиотеку
- Чтобы подключить библиотеку в чистом JS, нужно:

- Скачать js-файл с ней
- Положить файл в папку с проектом
- Подключить скрипт в html ДО скрипта, где собираетесь её использовать:

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, user-scalable=no, initial-sc
ale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <script src="js.cookie.min.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

- Чтобы установить новую пару ключ-значение с помощью библиотеки, нужно использовать объект Cookies и метод set:

```
Cookies.set('key', 'value');
```

- Чтобы получить значние по ключу с помощью библиотеки, нужно использовать объект Cookies и метод get:

```
const cookieValue = Cookies.get('key');
```