Обработка данных

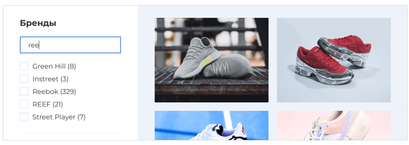
Практически любой JavaScript-код не обходится без обработки коллекций и массивов. Например, вы получили коллекцию DOM-элементов, её нужно в любом случае перебрать и при переборе обработать. Или с сервера пришёл массив банковских терминов для раздела Глоссарий и данные нужно сгруппировать по начальной букве перед выводом и так далее.

Разберём несколько полезных методов для обработки данных.

Группа перебирающих методов массива:

filter — метод создаёт новый массив, который содержит определённый набор данных из изначально заданного массива.

Метод хорошо подходит, например, для формирования выпадающего списка брендов товаров при вводе символов в поле Бренд.



Фильтрация списка брендов

Чтобы каждый раз не отправлять запрос на сервер, можно фильтровать данные прямо на клиенте.

Например, так:

```
const brands = ['Adidas', 'Green Hill', 'Instreet', 'Reebok', 'REEF', 'Street Player', ...];

const input = document.querySelector('.input');
const text = input.value;
const list = document.querySelector('.brands');

// фильтруем список брендов
// на всякий случай всё приводим к одному регистру (верхнему),
// чтобы поиск сделать регистронезависимым
// формируем HTML строку из тегов li
// в содержимое тега li - название бренда
const itemsString = brands
  .filter((value) => value.toUpperCase().includes(text.toUpperCase()))
  .map((item) => `<li>${item}</li>`)
  .join('');

list.insertAdjacentHTML('beforeend', itemsString);
```

О работе с DOM и добавлении элементов с помощью `insertAdjacentHTML` будет рассказано позже в углублённой теории.

find — метод для поиска в массиве, вернёт первый элемент, который удовлетворяет условию. Условие передаётся в виде *callback* функции, например, так:

```
const balls = [
  {
    color: 'red',
  },
  {
    color: 'blue',
  },
  {
    color: 'yellow',
  },
  {
    color: 'aqua'
  }
];


const yellowBall = balls.find((item) => item.color === 'yellow');
```

В результате из массива шаров будет выбран шар, у которого свойство `color` равно строке `'yellow'`, если таких шаров нет, то вернётся значение `undefined`. В нашем случае шар будет найден: `{color: 'yellow'}`.

includes — определяет, содержит ли массив определённый элемент, и в зависимости от этого возвращает `true` или `false`.

```
const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// вернёт true

console.log(pets.includes('raccoon'));
// вернёт false
```

forEach — метод для перебора массива. В современных браузерах с помощью `forEach` можно перебрать элементы коллекций *HTMLCollection* и *NodeList*.

```
const fields = document.querySelectorAll('.field');

// код работает только в современных браузерах
fields.forEach((field) => {
  field.toggleAttribute('disabled');
});
```

О DOM-элементах и коллекциях будет рассказано позже в углублённой теории.

map — метод создаёт новый массив на основе «старого». А функция в параметре метода описывает как будет видоизменятся каждый элемент исходного массива. Разберём этот метод на классическом примере с подсчётом длины слов в массиве.

```
// исходный массив имён
const rareNames = ['Аполлинария', 'Виолена', 'Илиирика', 'Цецилия', 'Янина'];

// новый массив, в котором элементы - это количество символов в имёни
const rareNamesLength = rareNames.map((item) => item.length);

console.log(rareNamesLength);
// [11, 6, 8, 7, 5]
```

join — метод для объединения всех элементов массива в одно строковое значение.

```
const strings = ['Видели ли вы меня в инвизибле?',
 'Из инвизибла я б не вылез бы.'];

// объединим строки в одну, при объединении между строками добавим пробел
console.log(strings.join(' '));
// "Видели ли вы меня в инвизибле? Из инвизибла я б не вылез бы."
```

Рассмотрим ещё один пример с использованием методов `map` и `join` в одной задаче. Добавим на страницу список товаров. Список будет дочерним элементом контейнера `<div class="container">`. В качестве исходных данных приходит массив вида:

```
const cards = [
 {
  title: 'Набор впечатлений «Счастливое мгновение»',
  img: 'img/plc-23.jpg',
  price: 1590,
  description: 'Набор из нескольких впечатлений. Получатель подарка сам выберет одно понравившееся
впечатление и воспользуется им в удобное для себя время!',
 },
 {...},
 ...
];
```

Сформируем HTML для списка товаров. С помощью метода `map` будем перебирать исходные данные и формировать разметку для элементов списка. А после всё соберём в одну строку, используя метод `join`. Итак, решение:

```
const container = document.querySelector('.container');

container.innerHTML = '';

// формируем строку с HTML для списка
const catalogString = `<ul class="catalog">
   ${cards.map(({title, img, price, description}) => `
    <li class="author-item">
      <article class="card">
        <img src='img/${img}" width="95" height="130" alt="${title}">
        <h3>${title}</h3>
        <p class="price">${price} <a class="cart" href="#">Купить</a></p>
        <p class="description short">${description}</p>
      </article>
    </li>
  `).join('')}
 </ul>`
);

container.insertAdjacentHTML('beforeend', catalogString);
```

Object.entries — метод возвращает массив собственных перечисляемых свойств указанного объекта в формате `[key, value]`. Затем этот массив можно перебрать в цикле `for...of`. Например так:

```
const params = {themeName: 'dark', currentPage: '768900'};

for (const [key, value] of Object.entries(params)) {
  console.log(key, value);
}
// "themeName" "dark"
// "currentPage" "768900"
```

Object.keys — когда нужны только ключи.

```
const params = {themeName: 'dark', currentPage: '768900'};

for (const key of Object.keys(params)) {
  console.log(`ключ: ${key}`);
}

// ключ: themeName , ключ: currentPage
```

Object.values — когда нужны только значения.

```
const params = {themeName: 'dark', currentPage: '768900'};

for (const value of Object.values(params)) {
  console.log(`значение: ${value}`);
}

// значение: dark , значение: 768900
```

Ознакомились со статьёй?

Сохранить прогресс

Дополнительные возможности синтаксиса JavaScript | Использование сторонних библиотек