

JavaScript в браузере

Сбербанк. Фронтенд школа.

JavaScript в браузере



WEB API

JavaScript в браузере. Web API.

- API для работы с документами, загруженными в браузер (DOM)
- API, принимающие данные от сервера (AJAX)
- API для работы с графикой (Canvas, WebGL)
- API для работы с аудио и видео (HTMLMediaElement, Web Audio API, и WebRTC)
- API устройств
- API хранения данных на стороне пользователя (WEB Storage, IndexedDB)

JavaScript в браузере. Web API.

Как работают API?

- Основаны на объектах (содержат в себе контейнеры для информации (свойства объекта) и реализуют функционал (методы объекта)

```
const coffeevarka = {  
  makeCoffee: function(coffee){  
    console.log("Начинаю готовить " + coffee);  
    setTimeout(()=>{console.log(coffee+ ' ГОТОВ')}, 3000)  
  },  
  
  drinks: ["латте", "капучино", "американо"],  
  
}  
  
const nespresso = Object.create(coffeevarka)  
const makeCoffeeButton = nespresso.makeCoffee('latte');  
//Начинаю готовить латте  
//Через три секунды сообщит, что латте готов  
const recipesButton = console.log(coffeevarka.drinks);  
//Выдает все возможные напитки
```

JavaScript в браузере. Web API.

Как работают API?

- У них узнаваемые точки входа (*nespresso(coffeevarka)* в предыдущем примере)
- Они используют события для управления состоянием (человек нажал кнопку в прошлом примере)

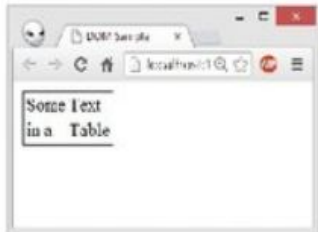
DOM

JavaScript в браузере. DOM.

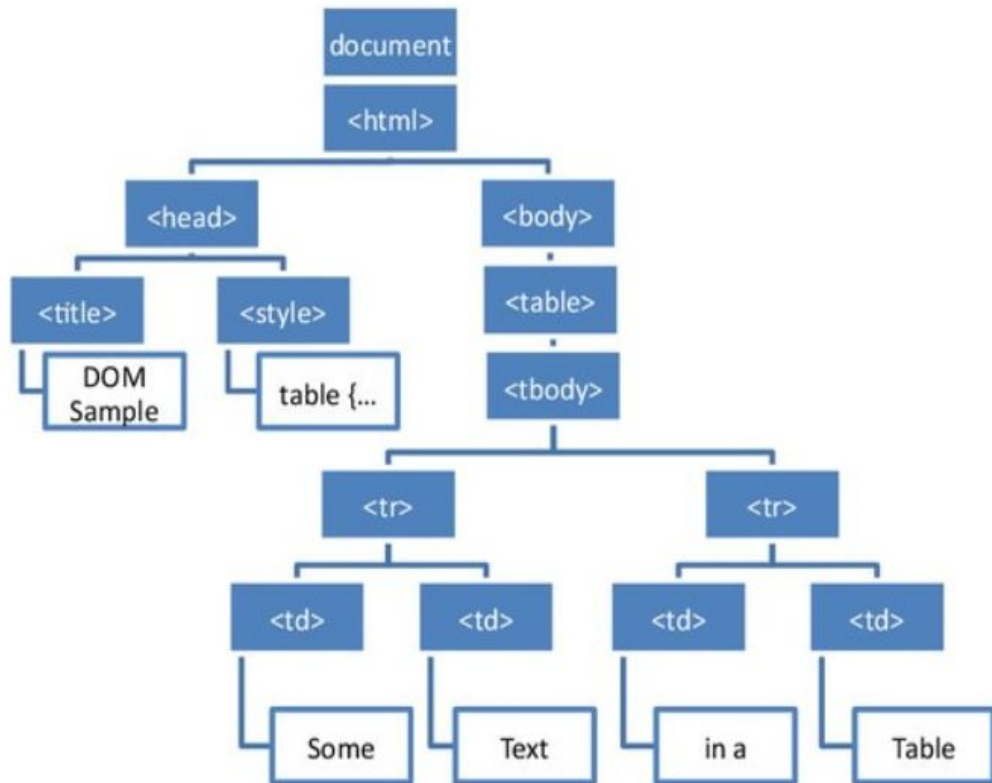
Как происходит процесс рендеринга страницы

1. Браузер получает HTML от сервера
2. Браузер формирует DOM (Document Object Model)
3. Загружаются и распознаются стили, формируется CSSOM (CSS Object Model).
4. На основе DOM и CSSOM формируется дерево рендеринга, или render tree — набор объектов рендеринга (сюда не попадают невидимые элементы)
5. Рассчитывается расположение всех элементов на странице (Flow / Layout)
6. Отрисовка (Painting)

DOM Tree



```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Sample</title>
    <style type="text/css">
      table {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <table>
      <tbody>
        <tr>
          <td>Some</td>
          <td>Text</td>
        </tr>
        <tr>
          <td>in a</td>
          <td>Table</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```



JavaScript в браузере. DOM.

Частоиспользуемые Типы Узлов:

Element: html-элемент

Document: корневой узел html-документа

Comment: элемент комментария

Text: текст элемента

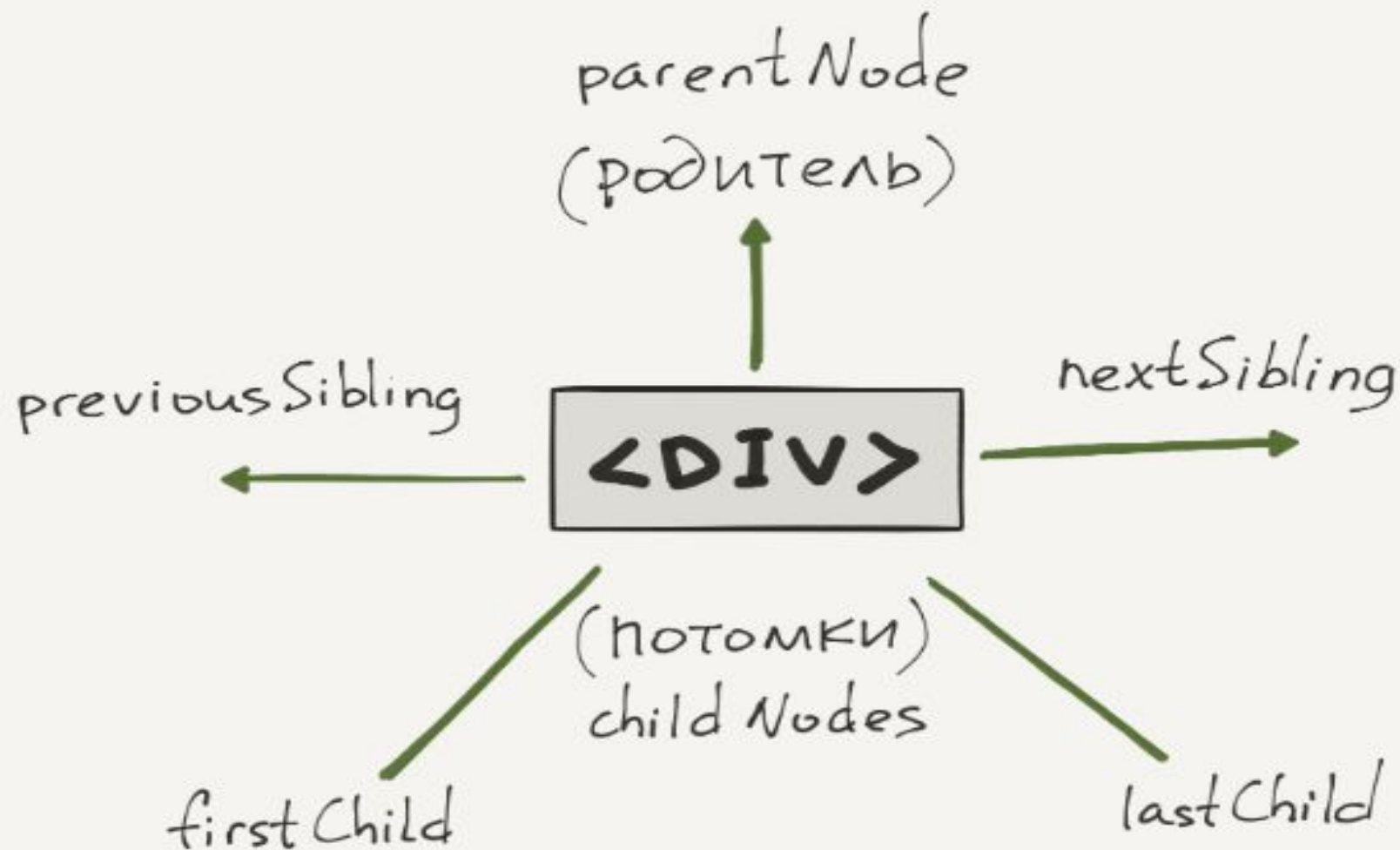
https://developer.mozilla.org/ru/docs/DOM/DOM_Reference#DOM_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D1%8B

JavaScript в браузере. DOM.

Все операции с DOM начинаются с объекта **document**. Это главная «точка входа» в DOM. Из него мы можем получить доступ к любому узлу.

Корневым элементом в DOM дереве считается элемент html. Доступ к нему можно получить так: **document.documentElement**.

```
const html = document.documentElement;  
// Read-only  
html.childNodes;  
html.firstChild;  
html.lastChild;  
html.childNodes[1];
```



JavaScript в браузере. DOM.

- **getElementById(value)**: выбирает элемент, у которого атрибут id равен value
- **getElementsByTagName(value)**: выбирает все элементы, у которых тег равен value
- **getElementsByClassName(value)**: выбирает все элементы, которые имеют класс value
- **querySelector(value)**: выбирает первый элемент, который соответствует css-селектору value
- **querySelectorAll(value)**: выбирает все элементы, которые соответствуют css-селектору value

JavaScript в браузере. DOM.

```
function changeColor(newColor) {  
  let elem = document.getElementById('first-article');  
  elem.style.color = newColor;  
}
```

```
<div class="month february">  
  <div class="week">Bill payed</div>  
  <div class="week"></div>  
  <div class="week"></div>  
  <div class="week"></div>  
</div>
```

```
document.getElementsByClassName('february').childNodes[1];  
document.querySelector(".month.february");
```

JavaScript в браузере. DOM.

Основные манипуляции

- 1) Создание и размещение новых узлов (*.createElement(el)*, *.appendChild(child)*, *.createTextNode(textNode)*)
- 2) Перемещение и удаление элементов (*.appendChild(child)*, *.removeChild(child)*)
- 3) Изменение стилей (*.style.attribute (color, textAlign, etc)*, *.setAttribute('class', newClassName)*)
- 4) Манипуляция классами (*.classList.add("mystyle") / remove*)

JavaScript в браузере. DOM.

Практика



<https://jsbin.com/toquxun/edit?html,css,js,output>

JavaScript в браузере. DOM.

Что должно примерно получиться



<https://jsbin.com/wihusac/1/edit?html,css,js,output>

JavaScript в браузере. DOM.

Почитать:

1. <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=ru>
2. <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=ru>
3. [https://developer.mozilla.org/ru/docs/Learn/JavaScript/Client-side web APIs/Manipulating documents](https://developer.mozilla.org/ru/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents)
4. <https://learn.javascript.ru/dom-navigation>

Событія

JavaScript в браузере. События

Events

- **События мыши** (перемещение курсора, нажатие мыши и т.д.)
- **События клавиатуры** (нажатие или отпускание клавиши клавиатуры)
- **События жизненного цикла элементов** (например, событие загрузки веб-страницы)
- **События элементов форм** (нажатие кнопки на форме, выбор элемента в выпадающем списке и т.д.)
- События, возникающие при изменении элементов DOM
- События, возникающие при касании на сенсорных экранах
- События, возникающие при возникновении ошибок

JavaScript в браузере. События

```
<html>
<head>
  <meta charset="utf-8" />
  <style>
    #rect{
      width:50px;
      height:50px;
      background-color:blue;
    }
  </style>
</head>
<body>
<div id="rect" onclick="handler(event)"></div>
<script>
function handler(e){

  alert(e.type); // получаем тип события
}
</script>
</body>
</html>
```

JavaScript в браузере. События

bubbles: возвращает true, если событие является восходящим. Например, если событие возникло на вложенном элементе, то оно может быть обработано на родительском элементе.

cancelable: возвращает true, если можно отменить стандартную обработку события

currentTarget: определяет элемент, к которому прикреплен обработчик события

defaultPrevented: возвращает true, если был вызван у объекта Event метод preventDefault()

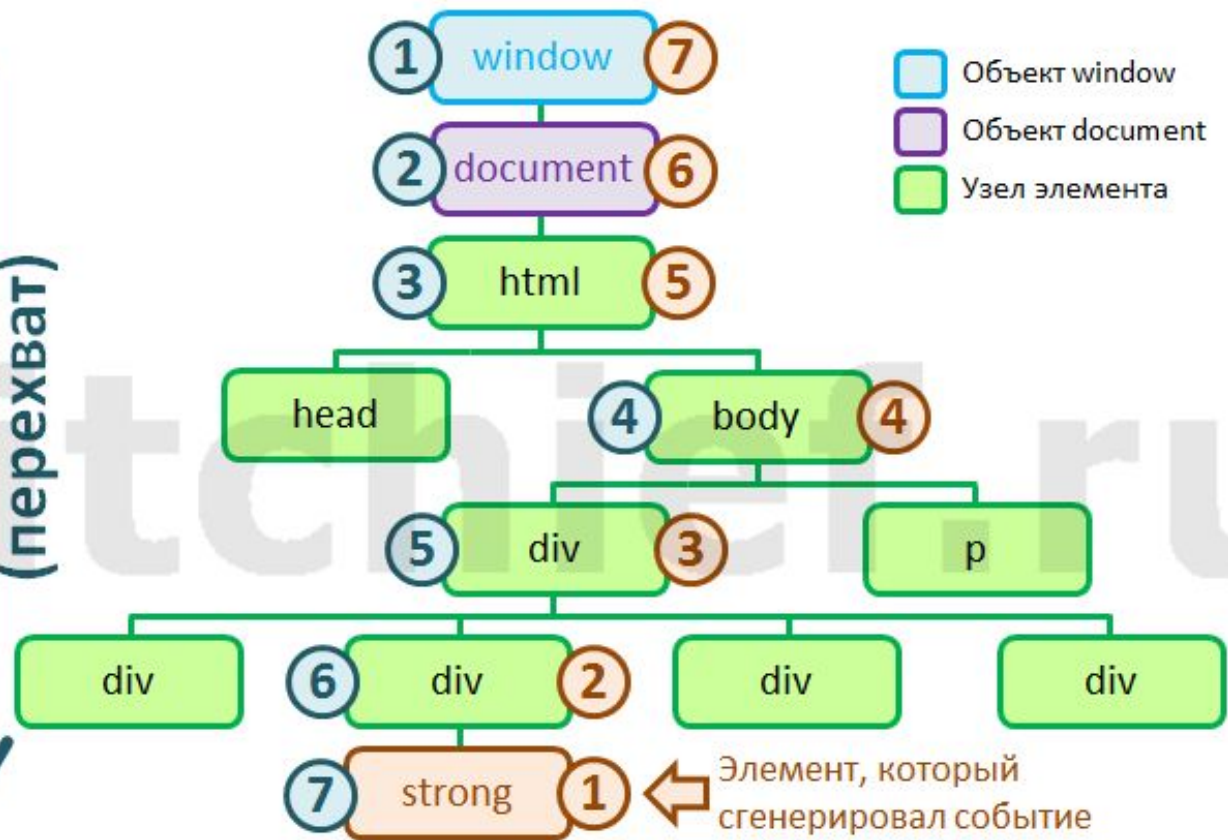
eventPhase: определяет стадию обработки события

target: указывает на элемент, на котором было вызвано событие

timeStamp: хранит время возникновения события

type: указывает на имя события

Этап 1: Погружение
(перехват)



Этап 3: Всплытие
события

Этап 2: Достижение цели, т.е. элемента
сгенерировавшего событие

JavaScript в браузере. События

Всплытие

Мы можем остановить всплытие событие с помощью метода `stopPropagation()` объекта `Event` (`e.stopPropagation()`)

Погружение события

Для их использования в метод `addEventListener()` в качестве третьего необязательного параметра передается логическое значение `true` или `false`, которое указывает, будет ли событие нисходящим. По умолчанию все события всплывающие.

<https://jsbin.com/xuqobey/edit?html,js,console,output>

JavaScript в браузере. События

Почитать

- 1) <https://itchief.ru/lessons/javascript/javascript-bubble-event>
- 2) <https://learn.javascript.ru/event-delegation>

Пытаешься Найти Бога?



Бог Ответит Тебе. Начни
Отношения С Ним Прямо
Сейчас



MirStudentov.com

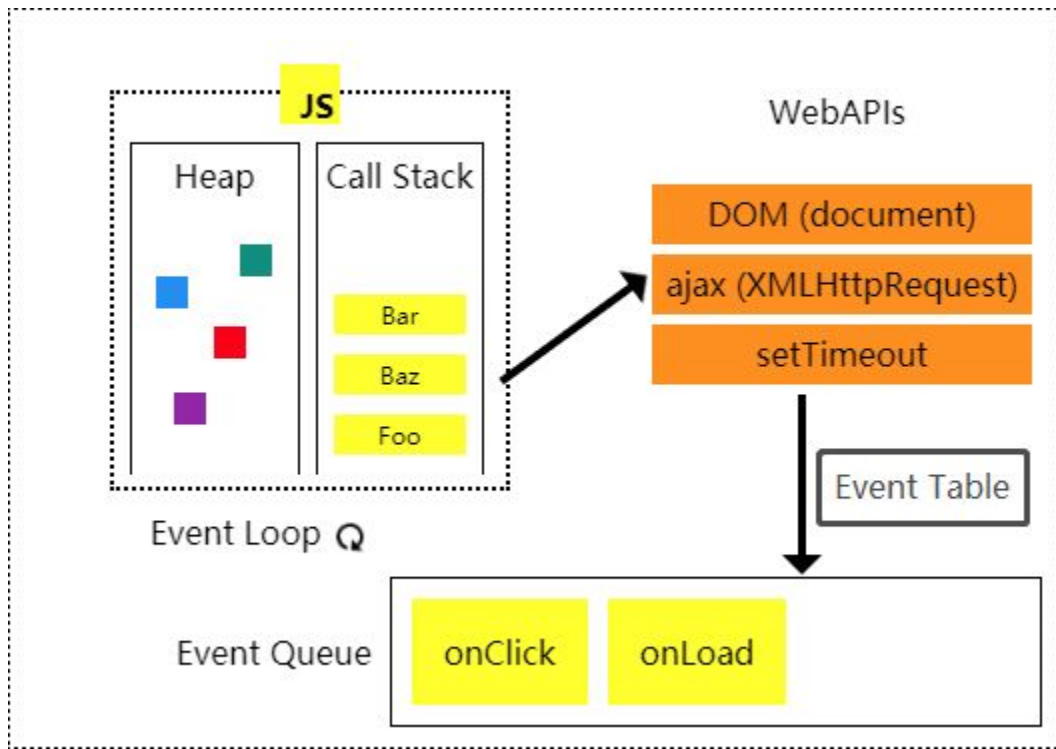
EventLoop

JavaScript в браузере. EventLoop

Внимание, вопрос:

```
setTimeout(()=>{console.log("First")}, 0);  
console.log("Second");
```

JavaScript в браузере. EventLoop





```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack


main()



```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack


main()



```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack

main()




```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack


main()


```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```



Call Stack

main()



```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(x) {  
  return multiply(x, x);  
}
```


```
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}
```

```
printSquared(4);
```

Call Stack

printSquared(4)

main()



```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(x) {  
  return multiply(x, x);  
}
```

```
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}
```

```
printSquared(4);
```

Call Stack

square(4)

printSquared(4)

main()



```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(x) {  
  return multiply(x, x);  
}
```

```
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}
```

```
printSquared(4);
```


Call Stack

multiply(4, 4)

square(4)

printSquared(4)

main()



```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(x) {  
  return multiply(x, x);  
}
```

```
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}
```


```
printSquared(4);
```

Call Stack

square(4)

printSquared(4)

main()




```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack

console.log(16)

printSquared(4)

main()



```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(x) {  
  return multiply(x, x);  
}
```


```
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}
```

```
printSquared(4);
```

Call Stack

printSquared(4)


main()



```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack

main()



```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(x) {  
  return multiply(x, x);  
}  
  
function printSquared(x) {  
  const squared = square(x);  
  console.log(squared);  
}  
  
printSquared(4);
```

Call Stack

JavaScript в браузере. EventLoop






```
console.log('foo');  
  
setTimeout(function () {  
  console.log('bar');  
}, 1000);  
  
console.log('baz');
```

Call Stack

main()



```
console.log('foo');  
  
setTimeout(function () {  
  console.log('bar');  
}, 1000);  
  
console.log('baz');
```

Call Stack

console.log('foo')


main()



```
console.log('foo');  
  
setTimeout(function () {  
  console.log('bar');  
}, 1000);  
  
console.log('baz');
```

Call Stack

main()



```
console.log('foo');  
  
setTimeout(function () {  
  console.log('bar');  
}, 1000);  
  
console.log('baz');
```

Call Stack

setTimeout(fn, 1000)

main()



```
console.log('foo');
```

```
setTimeout(function () {  
  console.log('bar');  
}, 1000);
```

```
console.log('baz');
```



Call Stack

main()



```
console.log('foo');
```

```
setTimeout(function () {  
  console.log('bar');  
}, 1000);
```

```
console.log('baz');
```

Call Stack

console.log('baz')

main()




```
console.log('foo');
```

```
setTimeout(function () {  
  console.log('bar');  
}, 1000);
```

```
console.log('baz');
```

Call Stack



```
console.log('foo');  
  
setTimeout(function () {  
  console.log('bar');  
}, 1000);  
  
console.log('baz');
```

Call Stack

Callback

timeout



```
console.log('foo');
```


```
setTimeout(function () {  
  console.log('bar');  
}, 1000);
```

```
console.log('baz');
```

Call Stack

**Callback
queue**

timeout



```
console.log('foo');  
  
setTimeout(function () {  
  console.log('bar');  
}, 1000);  
  
console.log('baz');
```

Call Stack

console.log('bar')

Callback
queue

JavaScript в браузере. EventLoop

- 1 поток, обрабатывающий Call Stack
- асинхронные события попадают в очередь Callback Queue
- Event Loop проверяет Call Stack на завершенность

JavaScript в браузере. EventLoop

Почитать

- 1) <https://www.knowledgescoops.com/2019/07/event-loops-event-tables-event-queues.html> (ENG)
- 2) <https://developer.mozilla.org/ru/docs/Web/JavaScript/EventLoop>

Асинхронность

JavaScript в браузере. Callbacks

Давным-давно...

```
function setInfo(name) {  
  address(myAddress) {  
    officeAddress(myOfficeAddress) {  
      telephoneNumber(myTelephoneNumber) {  
        nextOfKin(myNextOfKin) {  
          console.log('done');  
        };  
      };  
    };  
  };  
};
```


JavaScript в браузере. Promise

Promise

```
const promise1 = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});

promise1.then(function(value) {
  console.log(value);
  // expected output: "foo"
});

console.log(promise1);
// expected output: [object Promise]
```

JavaScript в браузере. Promise

Переходя на язык кофе

[https://jsbin.com/vagawuguwi/
edit?js,console](https://jsbin.com/vagawuguwi/edit?js,console)

JavaScript в браузере. Async / Await

- **Async/await** — это новый способ написания асинхронного кода. Раньше подобный код писали, пользуясь коллбэками и промисами.
- Конструкции, построенные с использованием `async/await`, как и промисы, **не блокируют главный поток выполнения программы**.
- Благодаря `async/await`, асинхронный **код становится похожим на синхронный**.

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/async_function

JavaScript в браузере. Async / Await

```
let coffeeIsReady = true

const promiseOfMakingCoffee =
  new Promise(function(resolve, reject) {
    if(coffeeIsReady){
      setTimeout(function() {
        resolve('Кофе готов!')
      }, 3000)
    } else {

      const reason = new Error('Бариста вылил на себя кипяток и его увезли в больницу')
      reject(reason)
    }
  });

const coffeeShop = (string) => console.log(string)

const makeOrder = async () => {
  const myOrder = await promiseOfMakingCoffee

  return myOrder
}

makeOrder().then(res => coffeeShop(res))
```

JavaScript в браузере. XMLHttpRequest (XHR)

XMLHttpRequest — это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.

```
let xhr = new XMLHttpRequest();
xhr.open('GET', '/my/url');
xhr.send();
xhr.onload = function() {
  if (xhr.status != 200) { // HTTP ошибка?
    // обрабатываем ошибку
    alert( 'Ошибка: ' + xhr.status);
    return;
  }
  // получим ответ из xhr.response
};
xhr.onprogress = function(event) {
  // выведем прогресс
  alert(`Загружено ${event.loaded} из ${event.total}`);
};

xhr.onerror = function() {
  // обрабатываем ошибку, не связанную с HTTP (например, нет соединения)
};
```

JavaScript в браузере. FETCH

FETCH — Стильно, модно, современно.

```
let promise = fetch(url, [options])
```

- url — URL для отправки запроса.
- options — дополнительные параметры: метод, заголовки и так далее.

Без options это простой GET-запрос, скачивающий содержимое по адресу url.

Браузер сразу же начинает запрос и возвращает промис, который внешний код использует для получения результата.

JavaScript в браузере. FETCH

Практика.

<https://jsbin.com/qejiwerine/edit?html,js,output>

JavaScript в браузере.

Почитать

1. <https://habr.com/ru/company/ruvds/blog/414373/> (async / await)
2. <https://medium.com/@stasonmars/%D0%BF%D1%80%D0%BE%D0%BC%D0%B8%D1%81%D1%8B-%D0%B2-javascript-%D0%B4%D0%BB%D1%8F-%D1%87%D0%B0%D0%B8%CC%86%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2-60bbef963541> (promise, async/await)
3. https://developer.mozilla.org/ru/docs/Web/API/Fetch_API/Using_Fetch (fetch)
4. <https://habr.com/ru/post/252941/> (fetch)