Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 4

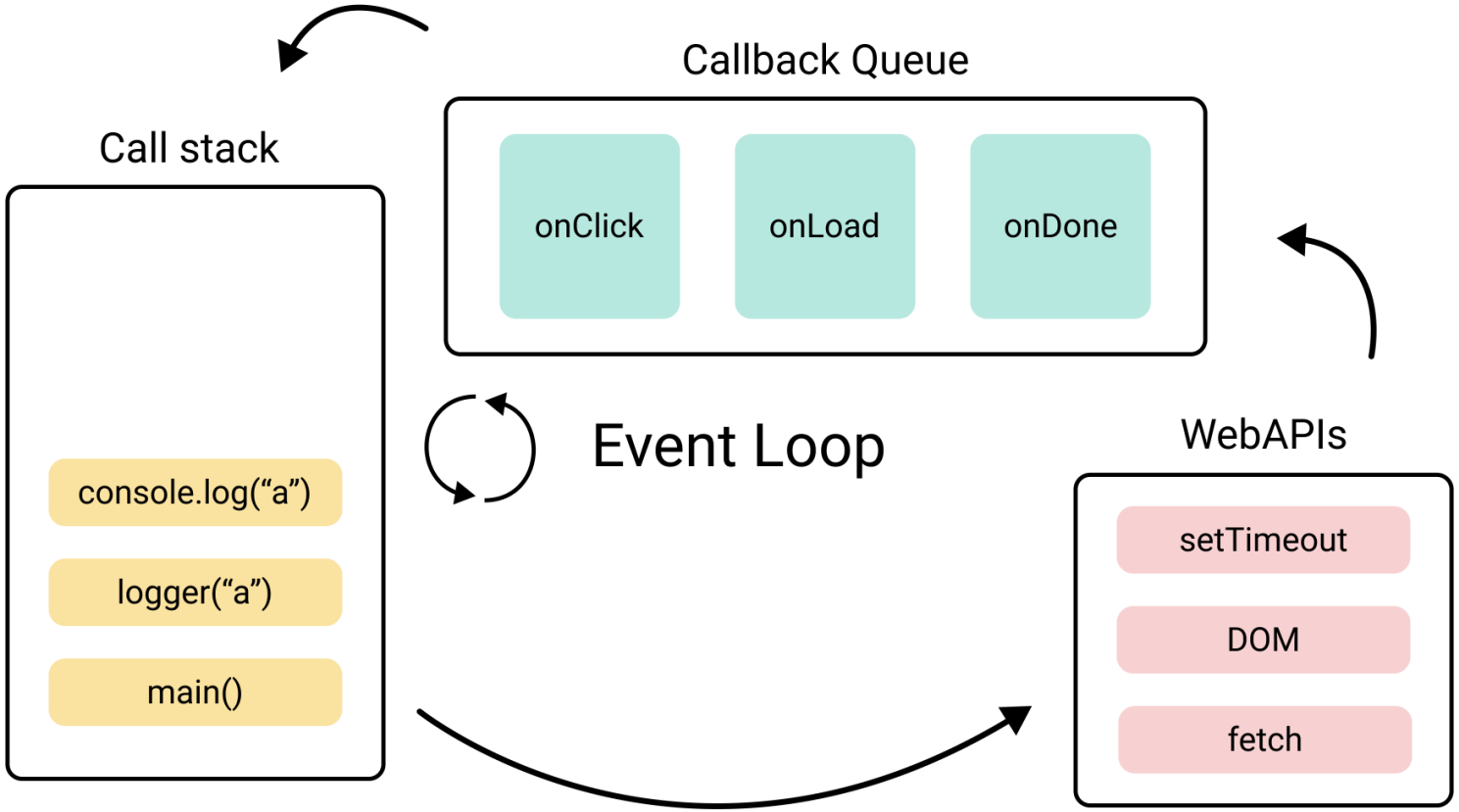
Асинхронный JS

План занятия

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. Event Loop – Цикл ожидания событий
2. Promises – Обещания
3. async / await
4. Fetch API

Event Loop


Callback Queue
onClick
onLoad
onDone
Call stack
console.log("a")
logger("a")
main()
Event Loop
WebAPIs
setTimeout
DOM
fetch

Promise

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Функции обратного вызова - эффективный способ обеспечить отложенное выполнение функции до тех пор, пока другая функция не завершится и не вернется с данными

Однако из-за вложенного характера обратных вызовов код может в конечном итоге запутаться, если у вас много последовательных асинхронных запросов, которые зависят друг от друга

Это было большим разочарованием для разработчиков JavaScript на раннем этапе, и в результате код, содержащий вложенные обратные вызовы, часто называют “пирамидой судьбы” или “адом обратного вызова” (callback hell)

```
function pyramidOfDoom() {
  setTimeout(() => {
    console.log(1)
    setTimeout(() => {
      console.log(2)
      setTimeout(() => {
        console.log(3)
      }, 500)
    }, 2000)
  }, 1000)
}
```

callbackhell.js

```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function() {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function() {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function() {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function() {
                                floppy.load('disk5', function (data5) {
                                    floppy.prompt('Please insert disk 6', function() {
                                        floppy.load('disk6', function (data6) {
                                            //if node.js would have existed in 1995
                                        });
                                    });
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});

```

Promise

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Промис - представляет собой асинхронную функцию

Это объект, который может вернуть значение в будущем, т.е. обещает вернуть

Выполняет ту же основную цель, что и функция обратного вызова, но с дополнительными функциями и более читаемым синтаксисом

Создание Промиса

```
// Initialize a promise
const promise = new Promise((resolve, reject) => {})
```

```
Output
__proto__: Promise
[[PromiseStatus]]: "pending"
[[PromiseValue]]: undefined
```

Promise


```
const promise = new Promise((resolve, reject) => {
  resolve('We did it!')
})

Output
__proto__: Promise
[[PromiseStatus]]: "fulfilled"
[[PromiseValue]]: "We did it!"
```

Асинхронные функции

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

```
const add = (a, b) => a + b
const res = add(2, 3)
console.log(res)

const add = (a, b, callback) => callback(a + b)
const res = add(2, 3, console.log)

const add = async (a, b) => {
    return a + b
}
const res = await add(2, 3)
```

Асинхронные функции

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Синтаксис async/await

```
const add = async (a, b) => {
    return a + b
}
const res = await add(2, 3)
console.log(res)
```

Асинхронные функции

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Синтаксис async/await вызова без await

```
const add = async (a, b) => {
    return a + b
}
const main = async () => {
    const res = await add(2, 3)
    console.log(res)
}

main()
```

Асинхронные функции

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Синтаксис async/await вызова с then

```
const add = async (a, b) => {
    return a + b
}

Распаковывает promise тут then
add(2, 3).then((res) => {
    console.log(res)
})
```

Асинхронные функции

Последовательное исполнение

```
const step1 = async () => console.log('step1')
const step2 = async () => console.log('step2')
const step3 = async () => console.log('step3')

await step1()
await step2()
await step3()
```

Если внутри асинхронной функции будет стоять setTimeout, то будем ждать пока он не выполниться

Асинхронные функции

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Последовательное исполнение через then

```
const step1 = async () => console.log(‘step1’)
const step2 = async () => console.log(‘step2’)
const step3 = async () => console.log(‘step3’)

await step1().then(step2).then(step3)
```

Асинхронные функции

Параллельное исполнение

```
const step1 = async () => console.log('step1')
const step2 = async () => console.log('step2')
const step3 = async () => console.log('step3')

step1()
step2()
step3()
```

Не забываем про таймеры в функциях, которые могут повлиять на время работы

Fetch API

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Fetch API - стандарт создания серверных запросов с промисами, также включающий много других возможностей

1. fetch(url)
2. .then(function() { })
3. .catch(function() { })

В целом, использование Fetch API выглядит

```
fetch(url)
.then(function() {

})
.catch(function() {

})
```

Fetch API

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Использование Fetch для получения данных от API

Следующие примеры кода основаны на Random User API

```
1. <ul id="authors"></ul>
2. <script>
  const ul = document.getElementById('authors')
</script>
3. const url = 'https://randomuser.me/api/?results=10’
4. function createNode(element) {      }
5. function createNode(element) {
       return document.createElement(element);
   }
6. function append(parent, el) {       }
7. function append(parent, el) {
       return parent.appendChild(el);
   }
```

Fetch API

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

8.
```
fetch(url)
  .then(function(data) {       })
  .catch(function(error) {     })
```
9.
```
fetch(url)
  .then((resp) => resp.json())
  .then(function(data) {
       let authors = data.results
       return authors.map(function(author) {  })
  })
```

Fetch API

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

```
function createNode(element) {
    return document.createElement(element);
}

function append(parent, el) {
  return parent.appendChild(el);
}

const ul = document.getElementById('authors');
const url = 'https://randomuser.me/api/?results=10';

fetch(url)
.then((resp) => resp.json())
.then(function(data) {
  let authors = data.results;
  return authors.map(function(author) {
    let li = createNode('li');
    let img = createNode('img');
    let span = createNode('span');
    img.src = author.picture.medium;
    span.innerHTML = `${author.name.first} ${author.name.last}`;
    append(li, img);
    append(li, span);
    append(ul, li);
  })
})
.catch(function(error) {
  console.log(error);
});
```

Fetch API

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

```
const url = 'https://randomuser.me/api';

let data = {
  name: 'Sara'
}

var request = new Request(url, {
  method: 'POST',
  body: data,
  headers: new Headers()
});

fetch(request)
.then(function() {
    // Handle response we get from the API
})
```

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание