

# Язык JavaScript, лекция 3

## Событийная модель Node.js

Пожидаев М. С.

21 октября 2022 г.

# Node.js

## *И его событийная модель*

1. Всё приложение разбивается на множество событий как можно меньшего размера.
2. События выполняются в одном общем потоке под управлением главного цикла (event loop).
3. Операция ввода-вывода обычно разбивается на событие, её запускающее, и событие, обрабатывающее её результат.

### Преимущества:

- ▶ можно не заботиться о согласовании потоков;
- ▶ возрастает общая эффективность исполнения.

# IO-bound

## *Неблокирующий ввод-вывод*

Текущий поток исполнения инициирует операцию ввода-вывода, после чего продолжает исполнение, откладывая обработку результата на потом. Получается намного эффективнее, но появляются две проблемы:

- ▶ что выполнить для продолжения работы;
- ▶ как и когда обработать результат.

Пример: DMA

# Минусы

## *Недостатки асинхронного программирования*

1. Нельзя смешивать блокирующие и неблокирующие вызовы.
2. Сильно затруднена обработка ошибок (использование исключений практически невозможно).

# Напомним!

## *Лямбда-выражения*

Лямбда-выражение — функция, сохраняемая и передаваемая как значение переменной.

1. Может использоваться для объявления вспомогательной функции в произвольном месте кода.
2. Следует тщательно следить за областью видимости переменных, с которыми ведётся работа внутри лямбда-выражения.

# Три стиля неблокирующих функций

1. Использование callback-функций.
2. Использование промисов.
3. Использование `async/await`.

# По-старинке

## *Пример вызова с callback-функцией*

```
this.objDir = (req, res, obj)=>{
  if (this.emptyStr(obj) || obj.trim().length < 8) {
    errors.internal(req, res);
    return;
  }
  const path = obj.trim();
  fs.mkdir(path, {recursive: true}, (e)=>{
    //Some code to handle the result
  });
};
```

# Промисы

*Обещание сделать работу потом*

Промис (promise) — заготовленный объект в JavaScript, задающий порядок отложенного выполнения кода. Может быть в одном из трёх состояний:

- ▶ pending;
- ▶ fulfilled;
- ▶ rejected.

Последующие действия задаются функцией (then()), которая может быть вызвана несколько раз (chaining).



# Создание промиса

```
function fooBar() {  
  return new Promise((resolve, reject)=>{  
    if (ok)  
      resolve('OK'); else  
      reject('Problem');  
  });  
}
```

# Вызов промиса

```
foobar()  
  .then((res)=>{})  
  ...  
  .catch((e)=>{});
```

# Промисифайзеры

Проблема промисов в том, что функция с получением результата через `callback` и функция, возвращающая промис, имеют два разных интерфейса. Поэтому появились так называемые промисифайзеры, которые автоматически анализируют структуру объекта и на каждую функцию с `callback` добавляют ещё одну, имеющую интерфейс промисифайзера. Очевидно, для работы промисифайзеры используют средства интроспекции языка.

Пример: `Bluebird` (добавляет функции с суффиксом `Async`).

# Ключевое слово `async`

Ключевое слово `async` используется при объявлении функции и означает, что:

1. Функция автоматически возвращает промис.
2. Внутри функции можно использовать ключевое слово `await`.
3. Функция не может выбрасывать исключения.

# Ключевое слово `await`

Ключевое слово `await` используется для вызова функции, возвращающей в качестве результата промис. При вызове:

1. Текущее исполнение функции прерывается, поток заполняется исполнением других событий в очереди.
2. Работа будет продолжена, когда промис перейдёт в состояние `fulfilled`.
3. Результат выполнения промиса будет возвращён как значение функции.

# Так делают сейчас

## *Пример асинхронной функции*

```
this.objDir = async (req, res, obj)=>{
  if (this.emptyStr(obj) || obj.trim().length < 8) {
    errors.internal(req, res);
    return;
  }
  const path = obj.trim();
  await fs.mkdirAsync(path, {recursive: true});
};
```

Спасибо за внимание!

Всё о курсе: <https://marigostra.ru/materials/js.html>

E-mail: [msp@luwrain.org](mailto:msp@luwrain.org)

Канал в Телеграм: <https://t.me/MarigostraRu>