

10. Дополнение:

Асинхронное логика и асинхронное программирование

10.1

Пример синхронного выполнения с объединением двух файлов через чтение и запись в другой файл с помощью синхронных функций `readFileSync` и `writeFileSync`, время выполнения кода = время чтения файла1 + время чтения файла2 + время записи нового файла:

```
import fs from 'fs';

const content1 = fs.readFileSync('./myfile1', 'utf-8');

const content2 = fs.readFileSync('./myfile2', 'utf-8');

fs.writeFileSync('./myfile-copy', content1 + content2 );
```

10.2

Пример асинхронного программирования. Задача объединения двух файлов с помощью асинхронных функций `readFile` и `writeFile` сводится к последовательному выполнению трех операций, так как записать новый файл мы можем лишь тогда, когда прочитаем данные первых двух. Упорядочить подобный код можно лишь одним способом: каждая последующая операция должна запускаться внутри `callback`'а предыдущей. Тогда мы построим нужную цепочку вызовов:

```
import fs from 'fs';

fs.readFile('./first', 'utf-8', (_error1, data1) => {
```

```

fs.readFile('./second', 'utf-8', (_error2, data2) => {

  fs.writeFile('./new-file', `${data1}${data2}`, (_error3) => {

    console.log('File has been written');

  });

});

});

```

10.3

Данный код гораздо эффективнее примера с синхронным чтением, так как во время чтения и записи файла процессор может выполнять другие команды, но его можно сделать еще более эффективным если читать оба исходных файла параллельно:

```

import fs from 'fs';

const state = {

  count: 0,

  results: [],

};

const tryWriteNewFile = () => {

  if (state.count !== 2) {

    return; // guard expression

  }

  fs.writeFile('./new-file', state.results.join(''), (error) => {

    if (error) {

      return;

    }

    console.log('finished!');

```

```
});  
  
};  
  
console.log('first reading was started');  
  
fs.readFile('./first', 'utf-8', (error1, data1) => {  
  
  console.log('first callback');  
  
  if (error1) {  
  
    return;  
  
  }  
  
  state.count += 1;  
  
  state.results[0] = data1;  
  
  tryWriteNewFile();  
  
});  
  
console.log('second reading was started');  
  
fs.readFile('./second', 'utf-8', (error2, data2) => {  
  
  console.log('second callback');  
  
  if (error2) {  
  
    return;  
  
  }  
  
  state.count += 1;  
  
  state.results[1] = data2;  
  
  tryWriteNewFile();  
  
});
```

Так как этот код асинхронный, результат работы каждой функции можно получить лишь внутри колбэков. Причём, порядок запуска колбэков мы не можем знать — всё зависит от того, какой файл прочитается быстрее. **Для отслеживания состояния выполнения этих операций придётся ввести глобальное состояние (относительно этих операций),**

через которое мы будем отслеживать завершенность и в котором сохраним данные.

И только когда все операции завершились — запишем новый файл. Кроме того, нам нужно чётко разделять данные первого и второго файлов, так как запись в новый файл (в отличие от чтения) должна происходить в определенном порядке.

Источник: [Асинхронное программирование. Параллельное выполнение операций.](#)

Приведенный код в примерах можно сделать гораздо более компактным и наглядным используя синтаксис оператора `async/await`, **но здесь специально выбран самый ранний синтаксис, чтобы подчеркнуть значение функции обратного вызова. Это гораздо более значимый феномен, чем рекурсия, которая оставляет нас в парадигме последовательного программирования архитектуры фон Неймана.**

10.4

Listener - прослушиватель:

Прослушиватель - это функция, которая связана с некоторым событием. Проще всего это продемонстрировать на примере пользовательского интерфейса в браузере:

```
<script>
```

```
function countRabbits() {  
  
  for(let i=1; i<=3; i++) {  
  
    alert("Кролик номер " + i);  
  
  }  
  
}
```

```
</script>
```

```
<input type="button" onclick="countRabbits()" value="Считать кроликов!">
```

В данном примере при каждом клике на кнопку вызывается функция которая выводит три сообщения о количестве кроликов. **Таким образом листенер - это тот же самый Callback, но он связан не с функцией, а с некоторым элементом, который обычно предназначен для связи в внешним миром.**

Источник: [Введение в браузерные события](#)

10.5

На web-сервере главные прослушиватели располагаются в модуле маршрутизации запросов (которые сервер получает извне).

```
const express = require('express'),

app = express();

const host = '127.0.0.1';

const port = 7000;

app.get('/api/users', (req, res) => {...});

app.post('/api/users', (req, res) => {...});

app.put('/api/users', (req, res) => {...});

app.delete('/api/users', (req, res) => {...});

app.listen(port, host, () => console.log(`Server listens http://\${host}:\${port}`))
```

Версия #3

GRN создал 28 July 2022 18:56:02

GRN обновил 28 July 2022 18:59:45