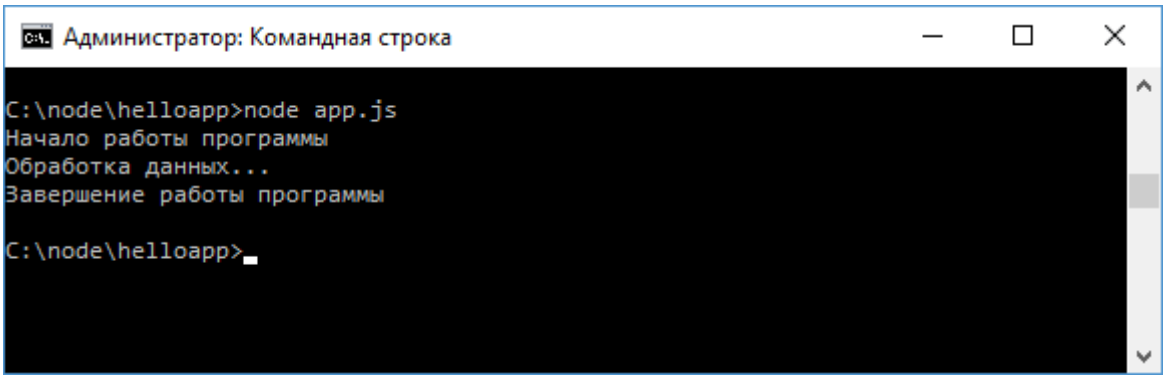
Асинхронність у Node.js

Асинхронність є можливість одночасно виконувати відразу кілька завдань. Асинхронність відіграє велику роль у Node.js.

Наприклад, у файлі програми *app.js* у нас розташований наступний код:

```
function displaySync(data){
    console.log(data);
}

console.log("Начало работы программы");

displaySync("Обработка данных...");

console.log("Завершение работы программы");
```

Це стандартний синхронний код, всі виклики тут виконуються послідовно, що ми можемо побачити, якщо ми запустимо програму:



Для розгляду асинхронності змінимо код файлу *app.js* таким чином:

```
function display(data, callback){

    // с помощью случайного числа определяем ошибку
    var randInt = Math.random() * (10 - 1) + 1;
    var err = randInt>5? new Error("Ошибка выполнения. randInt больше 5"): r
```

```
6
7       setTimeout(function(){
8           callback(err, data);
9       }, 0);
10  }
11
12  console.log("Начало работы программы");
13
14  display("Обработка данных...", function (err, data){
15
16      if(err) throw err;
17      console.log(data);
18  });
19
20  console.log("Завершение работы программы");
```

На початку також визначається функція display, але тепер крім даних як другий параметр вона приймає функцію зворотного виклику, яка і обробляє дані.

Эта функция callback принимает два параметра - информацию об ошибке и собственно данные. Это общая модель функций обратного вызова, которые передаются в асинхронные методы - первым идет параметр, представляющий ошибку, а второй - данные.

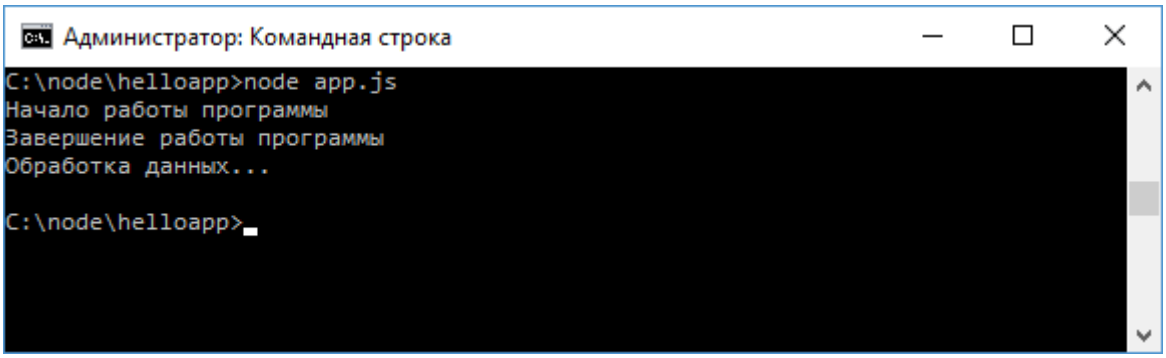
Для имитации ошибки используется случайное число: если оно больше 5, то создаем объект ошибки - объект Error, иначе же он равен null.

И последний важный момент - выполнение функции обратного вызова в функции **setTimeout()**. Это глобальная функция, которая принимает в качестве первого параметра функцию обратного вызова, а в качестве второго - промежуток, через который функция обратного вызова будет выполняться. Для нашей задачи вполне подойдет промежуток в 0 миллисекунд.

При вызове функции display в нее передается функция, которая в случае отсутствия ошибок просто выводит данные на консоль:

```
1  display("Обработка данных...", function (err, data){
2
3      if(err) throw err;
4      console.log(data);
5  });
```

Теперь если мы запустим приложение, то увидим, следующую картину:



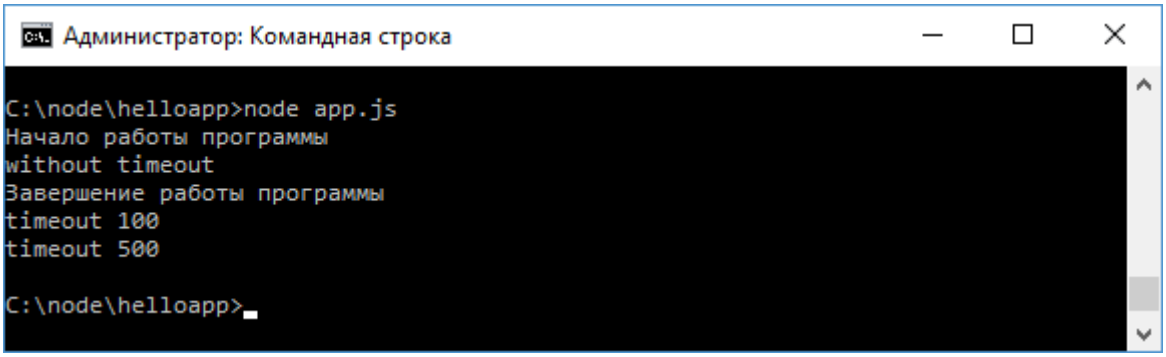
Несмотря на то, что в setTimeout передается промежуток 0, фактическое выполнение функции display завершается после всех остальных функций, которые определены в программе. В итоге выполнение на функции display не блокируется, а идет дальше. И это особенно актуально, если в приложении идет какая-либо функция ввода-вывода, например, чтения файла или взаимодействия с базой данных, выполнение которой может занять продолжительное время. То общее выполнение приложение не блокируется, а идет дальше.

Почему так происходит? Потому что все колбеки или функции обратного вызова в асинхронных функциях (в качестве таковой здесь используется функция setTimeout) помещаются в специальную очередь, и начинают выполняться после того, как все остальные синхронные вызовы в приложении завершат свою работу. Собственно поэтому выполнение колбека из функции setTimeout в примере выше происходит после выполнения вызова `console.log("Завершение работы программы");`. И стоит подчеркнуть, что в очередь колбеков переходит не функция, которая передается в display, а функция, которая передается в setTimeout.

Рассмотрим пример с двумя асинхронными вызовами:

```
function displaySync(callback){
    callback();
}

console.log("Начало работы программы");

setTimeout(function(){

        console.log("timeout 500");
}, 500);

setTimeout(function(){

        console.log("timeout 100");
}, 100);

displaySync(function(){console.log("without timeout")});
console.log("Завершение работы программы");
```

Результат выполнения:



Несмотря на то, что в функцию display передается колбек, но эта функция с колбеком будет выполняться синхронно.

А колбеки з функцій setTimeout будуть виконуватися лише після решти всіх викликів програми.

Назад **Зміст** **Вперед**