**Асинхронность в Node.js и Event Loop**

📘 Это один из ключевых модулей Node.js. От понимания асинхронности зависит, насколько эффективно ты сможешь строить масштабируемые серверные приложения.

**📚 1. Что такое Асинхронность?**

**Асинхронность** — это возможность выполнения операций без блокировки основного потока.

В Node.js это жизненно важно, потому что:

* Node работает **в одном потоке**
* Он использует **Event Loop** для управления задачами
* Ввод/вывод (I/O), сетевые запросы и таймеры обрабатываются **неблокирующе**

**⚙️ 2. Event Loop — сердце Node.js**

Когда ты запускаешь Node-программу, она не создаёт новые потоки (в отличие от Java или Python), а обрабатывает задачи через **Event Loop**.

**📊 Фазы Event Loop:**

1. **Timers** — обработка setTimeout() и setInterval()
2. **Pending Callbacks** — отложенные системные коллбэки
3. **Idle, Prepare** — внутренние процессы
4. **Poll** — опрос очереди I/O-событий
5. **Check** — выполнение setImmediate()
6. **Close callbacks** — завершение сокетов, файлов и т.п.

**💡 3. Асинхронные механизмы в Node.js**

**🔹 3.1. setTimeout() — планирует задачу**

js

КопироватьРедактировать

console.log("Начало");

setTimeout(() => {

console.log("Асинхронно: через 2 секунды");

}, 2000);

console.log("Конец");

🧵 **Объяснение:**

1. console.log("Начало") — выполняется синхронно.
2. setTimeout(...) — ставится в очередь.
3. console.log("Конец") — сразу выполняется.
4. Через 2 секунды Event Loop извлекает callback из очереди и выполняет.

**🔹 3.2. setImmediate() vs setTimeout()**

js

КопироватьРедактировать

setTimeout(() => {

console.log("setTimeout");

}, 0);

setImmediate(() => {

console.log("setImmediate");

});

🧵 **Разбор:**

* setTimeout(..., 0) — не совсем ноль. Она ждёт окончания текущей фазы Event Loop.
* setImmediate() выполняется **на следующей итерации** Event Loop.
* Иногда setImmediate вызывается раньше, иногда позже — не гарантировано.

**🔹 3.3. process.nextTick()**

js

КопироватьРедактировать

process.nextTick(() => {

console.log("nextTick");

});

console.log("Скрипт");

🧵 **Разбор:**

* nextTick() всегда исполняется **до** любого другого асинхронного вызова, даже перед setTimeout(0) и setImmediate.

**🔄 4. Callback Hell**

**😰 Пример:**

js

КопироватьРедактировать

fs.readFile('file.txt', (err, data) => {

if (err) throw err;

fs.writeFile('copy.txt', data, (err) => {

if (err) throw err;

fs.appendFile('log.txt', 'Файл скопирован', (err) => {

if (err) throw err;

console.log("Все операции завершены");

});

});

});

🔍 **Проблема:** Вложенность, сложность отладки, сложно обрабатывать ошибки.

**🧭 5. Promises**

**📦 Пример:**

js

КопироватьРедактировать

function asyncOp() {

return new Promise((resolve, reject) => {

setTimeout(() => {

resolve("Готово!");

}, 1000);

});

}

asyncOp().then((result) => {

console.log(result);

});

🧵 **Разбор:**

1. Promise — объект, который обещает результат.
2. resolve() — завершилось успешно.
3. reject() — ошибка.
4. .then() — обработка результата.

**🚀 6. async/await**

**🔹 Пример:**

js

КопироватьРедактировать

function fetchData() {

return new Promise((resolve) => {

setTimeout(() => resolve("Данные получены"), 1500);

});

}

async function main() {

console.log("Запуск");

const data = await fetchData();

console.log("Результат:", data);

}

main();

🧵 **Разбор:**

* async function — позволяет использовать await.
* await "останавливает" выполнение до получения результата.
* Код становится **линейным** и читабельным.

**⚠️ 7. Обработка ошибок в async/await**

js

КопироватьРедактировать

async function risky() {

try {

const data = await fetchData();

console.log(data);

} catch (err) {

console.error("Ошибка:", err);

}

}

**🧪 10 Практических ЗАДАНИЙ (Асинхронность)**

| **№** | **Задание** |
| --- | --- |
| 1 | Выведи в консоль start, потом через setTimeout 2 сек. — end |
| 2 | Напиши скрипт, где nextTick выполняется перед setTimeout |
| 3 | Используй setImmediate и setTimeout(0) и выведи порядок |
| 4 | Сделай цепочку из 3-х Promise, каждый задерживает результат |
| 5 | Создай async функцию, которая ждёт 2 Promise |
| 6 | Преобразуй callback-функцию fs.readFile() в Promise |
| 7 | Напиши try/catch для async функции с ошибкой в Promise |
| 8 | Сделай Promise.race() между двумя setTimeout |
| 9 | Используй util.promisify для fs.readFile |
| 10 | Реализуй простую загрузку данных с API с задержкой через fetch и await |