**JavaScript Promises & Async/Await — Complete Reference**

# Table of Contents



---

## Introduction

Async programming in JS allows non-blocking execution of tasks such as file I/O, HTTP requests, database operations, and timers. Modern JS uses Promises and async/await instead of callbacks.

## What is a Promise?

A Promise represents the eventual result of an async operation.

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const success = true;
    if(success) resolve("Success!");
    else reject("Error!");
  }, 1000);
});
```

## Promise States

- Pending: Operation in progress
- Fulfilled: Operation succeeded
- Rejected: Operation failed

```
stateDiagram-v2
    [*] --> Pending
    Pending --> Fulfilled : resolve(value)
    Pending --> Rejected : reject(reason)
```

## Creating and Using Promises

```
promise.then(res => console.log(res)).catch(err => console.error(err));
```

Promise chaining:

```
fetchData()
  .then(data => processData(data))
  .then(result => console.log(result))
  .catch(err => console.error(err));
```

## Async/Await

```
async function fetchData() {
  try {
    const result = await somePromiseFunction();
    console.log(result);
  } catch(err) {
    console.error(err);
  }
}
```

## Error Handling

```
try {
  const res = await promise;
} catch(err) {
  console.error(err);
}
```

## Loops and Async

### for...of (sequential)

```
for(const file of files){
  const content = await fs.promises.readFile(file, 'utf-8');
}
```

### forEach (does NOT wait)

```
files.forEach(async file => {
  const content = await fs.promises.readFile(file, 'utf-8');
});
```

## Sequential vs Parallel Execution

Sequential:

```
for(const file of files){
  await fs.promises.readFile(file, 'utf-8');
}
```

Parallel:

```
await Promise.all(files.map(f => fs.promises.readFile(f, 'utf-8')));
```

With error handling:

```
const results = await Promise.allSettled(files.map(f =>
fs.promises.readFile(f, 'utf-8')));
results.forEach(r => console.log(r.status, r.value || r.reason));
```

## Promise Utilities

- Promise.all([...])
- Promise.allSettled([...])
- Promise.race([...])
- Promise.any([...])
- Promise.resolve(value)
- Promise.reject(reason)

## Async/Await in Express.js

```
app.get('/', async (req, res) => {
  try {
    const data = await fs.promises.readFile('data.txt', 'utf-8');
    res.send(data);
  } catch(err) {
    res.status(500).send(err.message);
  }
});
```

## Edge Cases & Gotchas

1. await in forEach ignored
2. Sequential vs parallel execution
3. Promise.all fails if one rejects
4. Async functions always return Promises
5. Top-level await only in ESM
6. Mixing callbacks with async can fail silently

## Mixing Callbacks and Promises

```
// Bad
fs.readFile('file.txt', async (err, data) => {
  const result = await asyncFunc(data); // might fail silently
});

// Good
const data = await fs.promises.readFile('file.txt','utf-8');
const result = await asyncFunc(data);
```

## Best Practices

1. Use Promises + async/await over callbacks
2. Wrap async code in try/catch
3. Use for...of for sequential operations
4. Use Promise.all/allSettled for parallel
5. Only one res.send per Express route
6. Handle rejected Promises
7. Convert callbacks to Promises if needed

## Visual Diagrams

### Async Flow

```
flowchart TD
A[Start] --> B{Promise Pending?}
B -->|Yes| C[Wait]
B -->|No| D[Resolved]
C --> D
D --> E{Error?}
E -->|Yes| F[Catch block]
E -->|No| G[Continue]
```

### Sequential vs Parallel

Sequential:

```
sequenceDiagram
participant F1 as File1
participant F2 as File2
participant Main as MainThread
Main->>F1: readFile(F1)
F1-->>Main: data1
Main->>F2: readFile(F2)
F2-->>Main: data2
```

Parallel:

```
sequenceDiagram
participant F1 as File1
participant F2 as File2
participant Main as MainThread
Main->>F1: readFile(F1)
Main->>F2: readFile(F2)
F1-->>Main: data1
F2-->>Main: data2
```

## Cheat Sheet

| Pattern | Example | Notes |
|---|---|---|
| Sequential | for...of + await | Each iteration waits |
| Parallel | Promise.all([...]) | Runs concurrently |
| Parallel with errors | Promise.allSettled([...]) | Individual error handling |
| Express | async (req,res)=>{try{await}catch(err) {res.status(500)}} | Safe pattern |
| Top-level await | await fs.promises.readFile(...) | Only in ESM |
| Returning values | async ()=>value | Always returns Promise |
| Mixing callbacks | Avoid inside async | Use fs.promises |

## References


- MDN: Promises
- MDN: async/await
- Node.js fs.promises
- Promise.allSettled
- JavaScript.info: Async


---

End of Notes.