Advance Concept Of Node Js

Date: Page:

1.- Asynchronous Programming

- Callbacks: functions passed as arguments to be executed later.

```
function fetchData(callback) {
    setTimeout(() => {
        callback('data received');
    }, 1000);
}
fetchData((data) => {
    console.log(data);
});
```

- Promises: Represent eventual completion or failure of an async ops.

```
const fetchData = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('data received');
    }, 1000);
});
fetchData.then(data => {
    console.log(data);
}).catch(error => {
    console.error(error);
});
```

- Async/Await
- Syntactic code

```
async function fetchData() {
    try {
        const data = await fetchDataPromise();
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
fetchData();
```

Date: Page:

2.- Streams

Handle reading/writing of data in a continuous fashion.

```
const fs = require('fs')
const readableStream = fs.createReadStream('file.txt');
readableStream.on('data', (chunk) => {      // read
    console.log(chunk);
});
const writableStream = fs.createWriteStream('file.txt');
writableStream.write('Hello, world!');      // write
```

3.- Event loop

Core of Node.js runtime, handles non-blocking I/O operations.

```
setTimeout(() => {
    console.log('Timer!');
}, 0);
setImmediate(() => {
    console.log('Immediate!');
});
console.log('End!');
```

4.- Child Processes

Create subprocesses to perform tasks concurrently.

```
const {spawn} = require('child_process');
const ls = spawn('ls', ['-lh', '/usr']);
ls.stdout.on('data', (data) => {
    console.log(`stdout: ${data}`);
});
ls.stderr.on('data', (data) => {
    console.error(`stderr: ${data}`);
});
ls.on('close', (code) => {
```

```
    console.log(`child process exited with code ${code}`);
});
```

5. - Cluster Module

create multiple worker processes to handle load across multiple CPU cores.

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;
if (cluster.isMaster) {
    for (let i=0; i<numCPUs; i++) {
        cluster.fork();
    }
    cluster.on('exit', (worker, code, signal) => {
        console.log(`worker ${worker.process.pid} died`);
    });
} else {
    http.createServer((req, res) => {
        res.writeHead(200);
        res.end('Hello world\n');
    }).listen(8000);
}
```

6. - Debugging and Profiling

- Debugging - finding and fixing bugs
  node --inspect-brk app.js debuggers
- Profiling - Analyse performance
  node --prof app.js

7. - Security

- Use Helmet - Secure Express apps by setting HTTP headers.
- Input Validation - Prevent SQL Injection and XSS.
- Environment Variables -
- Regular Updates
- Es OWASP Recommendations

8. - Scalability and Performance Optimization

- load Balancing
- Caching
- Database Optimization
- Microservices Architecture
- Asynchronous Operations
- Use of Cluster Module
- Minimize Synchronous code
- Efficient Algorithms
- Avoid Memory leaks