

# Nodejs

- NodejsI/OWeb
- BFFSSRGraphQL
- WebpackGulpBabelJest

# Nodejs

Node.js Nodejs6

1. **timers**setTimeout()setInterval()
2. **I/O callbacks**I/OI/OI/O
3. **idle, prepare**Node.js
4. **poll**I/O
5. **check**setImmediate()
6. **close callbacks**socket

```
console.log('start');

setTimeout(() => {
  console.log('timeout');
}, 1000);

setImmediate(() => {
  console.log('immediate');
});

console.log('end');
```

```
bash
start
end
immediate
timeout
```

```
console.log('start');

setTimeout(() => {
  console.log('timeout');
  process.nextTick(() => {
    console.log('nextTick');
  });
}, 1000);

setImmediate(() => {
  console.log('immediate');
});

const fs = require('fs');
fs.readFile(__filename, () => {
  console.log('readFile');
  setImmediate(() => {
    console.log('immediate in readFile callback');
  });
  setTimeout(() => {
    console.log('timeout in readFile callback');
  }, 0);
});

process.nextTick(() => {
  console.log('nextTick');
});

console.log('end');
```

```
bash
start
end
nextTick
readFile
nextTick
immediate
immediate in readFile callback
timeout in readFile callback
timeout
nextTick
```

1. **timers** setTimeout    timeoutprocess.nextTick
2. **I/O callbacks** fs.readFile    readFilesetImmediatesetTimeout
3. **idle, prepare**
4. **poll** setImmediate    immediateimmediate in readFile    callbackfs.readFile  
setTimeout            timeout in readFile    callback
5. **check**    process.nextTick    nextTicknextTick

## 6. close callbacks

# EventEmitter

EventEmitterVueEventBusEventEmitter

Node.js EventEmitter httpnetfsEventEmittersocket.ionodemailercheerio

EventEmitter

NodejsEventEmitter

EventEmitteron

```
const fs = require('fs');

fs.readFile('file.txt', (err, data) => {
  if (err) {
    console.error(`Failed to read file: ${err}`);
  } else {
    console.log(`File content: ${data}`);
  }
});
```

EventEmitter

```
const fs = require('fs');

const stream = fs.createReadStream('file.txt');

stream.on('data', (chunk) => {
  console.log(`Received ${chunk.length} bytes of data.`);
});

stream.on('end', () => {
  console.log('Finished reading file.');
```

EventEmitter

# Buffer

Buffer 8

## 1. Buffer.from()net

```

const net = require('net');

const client = net.createConnection({ port: 8080 }, () => {
  //
  const data = Buffer.from('Hello, world!', 'utf8');

  //
  client.write(data);
});

```

## 2. Buffer

```

const fs = require('fs');

// Buffer
const data = fs.readFileSync('/path/to/file');

//
// ...

```

## 3. crypto

```

const crypto = require('crypto');

//
const key = Buffer.from('mysecretkey', 'utf8');
const iv = Buffer.alloc(16);

//
const cipher = crypto.createCipheriv('aes-256-cbc', key, iv);

//
const encrypted = Buffer.concat([cipher.update(data), cipher.final()]);

```

## 4.

```

const fs = require('fs');
const sharp = require('sharp');

// Buffer
const data = fs.readFileSync('/path/to/image');

//
sharp(data)
  .resize(200, 200)
  .toFile('/path/to/resized-image', (err, info) => {
    // ...
  });

```

# I/O

I/O/I/O/O

**I/O**

Nodejs I/O I/O Node.js

I/O

**I/O**

- I/O
- I/O
- CPU I/O I/O I/O

**CPU**

- CPU
- I/O

**Nodejs**

- **Koa** Nodejs API
- **Express** Express Express Koa Nodejs Koa Express
- **Eggjs** Koa Rest Restful API
- **Nestjs** TS, Java Springboot Nestjs GraphQL WebSocket MQ

**Koa**

Koa

1. middleware

```
use (fn) {  
  if (typeof fn !== 'function') throw new TypeError('middleware must be a  
function!')  
  debug('use %s', fn._name || fn.name || '-')  
  this.middleware.push(fn)  
  return this  
}
```

2. composePromise

```

function compose(middleware) {
  if (!Array.isArray(middleware)) throw new TypeError('Middleware stack must be an array!')
  for (const fn of middleware) {
    if (typeof fn !== 'function') throw new TypeError('Middleware must be composed of functions!')
  }

  return function (context, next) {
    let index = -1
    return dispatch(0)

    function dispatch(i) {
      if (i <= index) return Promise.reject(new Error('next() called multiple times'))
      index = i
      let fn = middleware[i]
      if (i === middleware.length) fn = next
      if (!fn) return Promise.resolve()
      try {
        return Promise.resolve(fn(context, dispatch.bind(null, i + 1)))
      } catch (err) {
        return Promise.reject(err)
      }
    }
  }
}

```

## Stream

Stream API

StreamAPI

## BFF

BFFBackend For Frontend,

BFFSSRGraphQLSSRSEOGraphQLAPI

## ORMNodejsORM

SQLORMSQL

Node.jsORM

1. **Sequelize**SequelizePromiseORMMySQLPostgreSQLSQLiteMicrosoft SQL Server API

2. **TypeORM**TypeORMTypeScriptORMMySQLPostgreSQLSQLiteMicrosoft SQL Server  
Oracle

## Redis

1. RedisRedis
- 2.
3. RedisRedis
4. RedisRedisRDBAOF
5. RedisRedisRedis

1. explainSQLSQL
2. Mysql
3. MySQL
- 4.
- 5.

id

NginxZookeeperDubboMQRPC

API

Spring CloudSpring Cloud Alibaba