Node.js 原理详解

Event Loop 事件循环模型

正常调用栈

```
function multiply(a, b) {
  return a * b;
}

function square(n) {
  return multiply(n, n);
}

function printSquare(n) {
  console.log(square(n));
}

printSquare(4);
```

调用栈出错

```
function multiply(a, b) {
  throw new Error();
}

function square(n) {
  return multiply(n, n);
}

function printSquare(n) {
  console.log(square(n));
}

printSquare(4);
```

调用栈溢出

```
function fn(a) {
  fn(a);
}

fn(1);
```

阻塞调用

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

fs.readFileSync('./a.txt');
fs.readFileSync('./b.txt');
fs.readFileSync(‘./c.txt');

console.log('finish')
```

异步调用

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

fs.readFile('./a.txt');
fs.readFile('./b.txt');
fs.readFile('./c.txt');

console.log('finish')
```

事件循环讲解

```
setTimeout(() => {
  console.log('timeout');
}, 5000)

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

宏任务与微任务

```
setTimeout(() => {
  console.log('timeout');
}, 0)

new Promise((resolve) => {
  console.log('promise');
  resolve();
}).then(() => {
  console.log('then');
})

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

Buffer

- Buffer 是 UInt8Array
- 是数组，且每个 item 有效范围是 0~255

```
Buffer.from([1, 1, 1, 1]);

Buffer.from([257, 257.5, -255, '1']);

Buffer.from('abcd');
// <Buffer 61 62 63 64>
```

Events

- on 方法，注册事件回调
- emit 方法，手动触发时间

```
const EventEmitter = require('events');

class MyEventEmitter extends EventEmitter {}
```

```
const myEventEmitter = new MyEventEmitter();

myEventEmitter.on('ping', function() {
  console.log('pong');
})

myEventEmitter.emit('ping');
```

Stream

- Node.js 中很多对象 都是 Stream, 例如 HTTP 的请求， 进程日志输出，文件的读写
- Stream 本身是一个 EventEmitter
- Stream 内部含有 Buffer
- 当 Stream 中 Buffer 有数据可读时，emit data 事件，通知外部读取数据
- 当 Stream 可写是，通过调用 write(), end() API 来写入数据到 内部 Buffer 中

Stream 的类型

- 可读 Writable
- 可写 Readble
- 双工 Duplex
- 转换 Transform

Node.js 全局对象

常用的全局对象

- clearInterval
- setInterval
- clearTimeout
- setTimeout
- console
- process

全局对象与模块

- __filename
- __dirname
- exports
- module
- require

这 5个 API 看上去像是 全局对象，但其实是在模块加载的时候进行注入，所以要和 全局对象 进行区分