## **1. yargs**

* [yargs](http://yargs.js.org/)模块能够解决如何处理命令行参数

### **1.1 安装**

cnpm install yargs --save

### **1.2 读取命令行参数**

* yargs模块提供了argv对象，用来读取命令行参数

hello.js

#!/usr/bin/env node**let** argv = require('yargs')

.alias('n', 'name')//还可以指定参数别名 -n的别名是--name

.demand(['n'])//是否必填

.default({ name: 'zhufeng' })//默认值

.describe({ name: '你的姓名' })// 参数描述

.boolean(['private'])

.argv;console.log(process.argv);console.log(argv);console.log('hello', argv.name);console.log(argv.\_);//argv对象有一个下划线属性，可以获取非连词线开头的参数

```js

node hello.js --private A B C

[

'C:\\Program Files\\nodejs\\node.exe',

'C:\\vipdata\\prepare6\\zhufeng\_loader\\5.js',

'--private',

'A',

'B',

'C'

]

{

\_: [ 'A', 'B', 'C' ],

private: true,

name: 'zhufeng',

n: 'zhufeng',

'$0': '5.js'

}

hello zhufeng

[ 'A', 'B', 'C' ]

#!/usr/bin/env node**let** argv = require('yargs')

.option('n',//参数n

{

alias: 'name',//别名name

demand: true,//必填

default: 'zhufeng',//默认值

describe: '你的姓名',//描述

type: 'string',//参数类型

}).usage('Usage: hello [options]')// 用法格式

.example('hello -n zhufeng', 'hello zhufeng')//示例

.help('h')//显示帮助 信息

.alias('h', 'help')//显示帮助信息

.epilog('copyright 2019')//出现在帮助信息的结尾

.argvconsole.log(process.argv);console.log(argv);console.log('hello', argv.name);console.log(argv.\_);//argv对象有一个下划线属性，可以获取非连词线开头的参数

```js

node hello.js -n jiagou

[

'C:\\Program Files\\nodejs\\node.exe',

'C:\\vipdata\\prepare6\\zhufeng\_loader\\6.js',

'-n',

'jiagou'

]

{ \_: [], n: 'jiagou', name: 'jiagou', '$0': '6.js' }

hello jiagou

[]

## **2. Semaphore**

* [this.semaphore](https://github.com/webpack/webpack/blob/v4.39.3/lib/Compilation.js" \l "L857-L971)
* [Semaphore.js](https://github.com/webpack/webpack/blob/94929a59a79bc79cab789804d5592c3ec0605cc4/lib/util/Semaphore.js)
* [Semaphore](https://www.npmjs.com/package/semaphore)
* Semaphore(信号量)控制并发访问量

### **2.1 使用Semaphore**

**let** Semaphore = require('semaphore');**let** semaphore = **new** Semaphore(2);console.time('cost');

semaphore.take(**function** () {

setTimeout(() => {

console.log(1);

semaphore.leave();

}, 1000);

});

semaphore.take(**function** () {

setTimeout(() => {

console.log(1);

semaphore.leave();

}, 2000);

});

semaphore.take(**function** () {

console.log(3);

semaphore.leave();

console.timeEnd('cost');

});

### **2.2 实现Semaphore**

**class** **Semaphore** {

**constructor**(available) {

**this**.available = available;

**this**.waiters = [];

**this**.\_continue = **this**.\_continue.bind(**this**);

}

take(callback) {

**if** (**this**.available > 0) {

**this**.available--;

callback();

} **else** {

**this**.waiters.push(callback);

}

}

leave() {

**this**.available++;

**if** (**this**.waiters.length > 0) {

process.nextTick(**this**.\_continue);

}

}

\_continue() {

**if** (**this**.available > 0) {

**if** (**this**.waiters.length > 0) {

**this**.available--;

**const** callback = **this**.waiters.pop();

callback();

}

}

}

}

### **2.3 webpack中的Semaphore**

class Semaphore {

constructor(available) {

this.available = available;

this.waiters = [];

this.\_continue = this.\_continue.bind(this);

}

+ acquire(callback) {

if (this.available > 0) {

this.available--;

callback();

} else {

this.waiters.push(callback);

}

}

+ release() {

this.available++;

if (this.waiters.length > 0) {

process.nextTick(this.\_continue);

}

}

\_continue() {

if (this.available > 0) {

if (this.waiters.length > 0) {

this.available--;

const callback = this.waiters.pop();

callback();

}

}

}

}

## **3. neo-async**

* [webpack\lib\Compilation.js](https://github.com/webpack/webpack/blob/v4.39.3/lib/Compilation.js" \l "L836)
* [neo-async](https://www.npmjs.com/package/neo-async)
* neo-async库和async库类似，都是为异步编程提供一些工具方法，但是会比async库更快

### **3.1 使用**

**let** {forEach} = require('neo-async');console.time('cost');**let** array = [1, 2, 3];

forEach(array, **function** (num, done) {

setTimeout(**function** () {

console.log(num);

done();

}, num \* 1000);

}, **function** (err) {

console.timeEnd('cost');

});

### **3.2 实现**

**function** **forEach**(array, iterator, callback) {

**let** length = array.length;

**function** **done**() {

**if** (--length == 0)

callback();

}

array.forEach((item, index) => {

iterator(item, done);

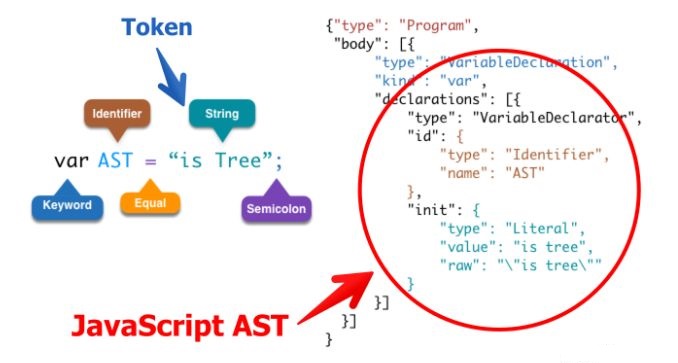
});

}

## **4.acorn**

### **4.1 介绍**

* [astexplorer](https://astexplorer.net/)可以把代码转成语法树
* [acorn](https://github.com/acornjs/acorn) A small, fast, JavaScript-based JavaScript parser
* acorn 解析结果符合[The Estree Spec](https://github.comb/estree/estree)规范(（这是 Mozilla 的工程师给出的 SpiderMonkey 引擎输出的 JavaScript AST 的规范文档[SpiderMonkey](https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Parser_API))



**const** ast = acorn.parse(code, options);

### **4.2 配置项 [#](https://zhufeng-document.vercel.app/html/75.webpack-5.prepare.html" \l "t124.2 %E9%85%8D%E7%BD%AE%E9%A1%B9)**

* ecmaVersion 你要解析的 JavaScript 的 ECMA 版本,默认是 ES7
* sourceType 这个配置项有两个值：module 和 script，默认是 script.主要是严格模式和 import/export 的区别.
* locations 默认值是 false，设置为 true 之后会在 AST 的节点中携带多一个 loc 对象来表示当前的开始和结束的行数和列数。
* onComment 传入一个回调函数，每当解析到代码中的注释时会触发，可以获取当年注释内容，参数列表是：[block, text, start, end],block 表示是否是块注释，text 是注释内容，start 和 end 是注释开始和结束的位置

### **4.2 查找依赖**

**const** acorn = require("acorn")**const** walk = require("acorn-walk");**const** escodegen = require('escodegen');**const** ast = acorn.parse(`

import $ from 'jquery';

let \_ = require('lodash');

`, { locations: true, ranges: true, sourceType: 'module', ecmaVersion: 8 });**let** resources = [];

walk.simple(ast, {

CallExpression(node) {

**if** (

node.type === 'CallExpression' &&

node.callee.type === 'Identifier' &&

node.callee.name === 'require' &&

node.arguments.length === 1 &&

node.arguments[0].type === 'Literal'

) {

**const** args = node.arguments;

resources.push({

module: args[0].value

})

}

},

ImportDeclaration(node) {

node.specifiers[0].local.name = 'jQuery';

resources.push({

module: node.source.value

})

}

})console.log('resources', resources);**let** result = escodegen.generate(ast);console.log(result