## 变量声明const和let

1. let于var

let和var都用于声明变量，但var会提升变量到根作用域，而let不会，如下示例

由于var变量的提升，所有实际代码是这样的：

function aa() {

    if (bool) {

        // 在这里定义 test

        var test = 'hello man'

    } else {

        // 输出 hello man

        console.log(test)

    }

}

如果我们将var换成let则会报错，因为let不提升变量

function aa() {

    if (bool) {

        // 在这里定义 test

        let test = 'hello man'

    } else {

        // 报错，找不到test

        console.log(test)

    }

}

1. const

const用于定义常量，定义之后不能改变

const name = 'lux'

name = 'joe' //再次赋值此时会报错

## 模板字符串

模板字符串就是``，如下示例

let name = '小陈';

let str = `你好，${name}`;

我们可以通过${}这样的方式来拼接字符串

**函数默认参数**

如下，当我们不传参数时num默认为200

function action(num = 200) {

    console.log(num)

}

action() //200

**箭头函数**

## 箭头函数就是一个函数

// 示例1

const add = (l, r) => l + r;

add(1, 2);

// 示例2，返回对象

const getDetail = () => ({

    a: 1,

    b: 2

});

箭头函数是运行时生成的，箭头函数的this指向的是运行时的this

## 解构

## 如下是数组和对象的解构，我们通过解构快速获取里面的变量

//对象

const people = {

    name: 'lux',

    age: 20

};

const { name, age } = people;

//数组

const color = ['red', 'blue'];

const [first, second] = color;

## Spread Operator 展开运算符

ES6的一个操作符 ...，其意思为展开对象

// 数组

const color = ['red', 'yellow']

// 展开color后我们得到 'red', 'yellow'，然后我们得到了一个新数组

// [red, yellow, green, pink]

const colorful = [...color, 'green', 'pink']

// 对象

const alp = { fist: 'a', second: 'b' }

// 展开后重组我们得到了一个新对象 { "fist": "a", "second": "b", "third": "c" }

const alphabets = { ...alp, third: 'c' }

当然，我们还可以这样操作

// 数组

const number = [1, 2, 3, 4, 5]

const [first, ...rest] = number

console.log(rest) // 输出为 2,3,4,5

// 对象

const user = {

    username: 'lux',

    gender: 'female',

    age: 19,

    address: 'peking'

}

const { username, ...rest } = user

console.log(rest) // 输出为 {"address": "peking", "age": 19, "gender": "female"}

## import 和 export

## export用于导出模块，import则用于导入模块

// 导出默认, 有且只有一个默认

export default {

    username: '',

    agx: 100

}

// 部分导出

export const other = {

    // ...

}

//导入部分

import person, { other } from './person'

## Promise

## Promise是异步编程的语法，语法太多，详细看其他章节

## Generators

生成器（ generator）是能返回一个**迭代器**的函数。生成器比其他函数多了一个星号。

// 生成器

function\* createIterator() {

    yield 1;

    yield 2;

    yield 3;

}

// 生成器能像正规函数那样被调用，但会返回一个迭代器

let iterator = createIterator();

for(let item of iterator) {

    // 1、2、3

    console.log(item);

}