

一、Node简介

1.1、为什么学习Node(了解)

企业需求 增加自身职业竞争 进一步理解 Web开发，并有助于明白后端开发 大前端必备技能 为了更好的学习前端框架

1.2、Node是什么

Node.js®是基于 Chrome的V8 JavaScript 引擎构建的JavaScript运行环境。

Node.js不是新语言，也不是一个框架或者一个库，而是一个软件。

Node.js是一个 JavaScript 运行环境，说白了，就是用来运行js的。

官网：<https://nodejs.org/en/> 中文官网：<https://nodejs.org/zh-cn/>

1.3、Node能做什么

Node 打破了过去 JavaScript 只能在浏览器中运行的局面。前后端编程环境统一，大大降低了前后端语言切换的代价。以下是Node可以实现的工作：（仅作参考）

- Web 服务器
- 命令行工具
- 网络爬虫
- 桌面应用程序开发（[Electron](#)）
- app
- 嵌入式
- 游戏
-

1.4、安装Node

官网：<https://nodejs.org/en/>

中文官网：<https://nodejs.org/zh-cn/>

Downloads

Latest LTS Version: 12.16.0 (includes npm 6.13.4)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Recommended For Most Users

Current
Latest Features


Windows Installer
node-v12.16.0-x64.msi


macOS Installer
node-v12.16.0.pkg


Source Code
node-v12.16.0.tar.gz

Windows Installer (.msi)
Windows Binary (.zip)
macOS Installer (.pkg)
macOS Binary (.tar.gz)
Linux Binaries (x64)
Linux Binaries (ARM)
Source Code

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v12.16.0.tar.gz	

win键+R键打开命令行窗口

```
C:\WINDOWS\system32\cmd.exe - node
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\VULCAN>node
Welcome to Node.js v12.13.1.
Type ".help" for more information.
>
```

输入node进入以上模式表示node安装成功。

二、使用Node执行js代码的两种方式

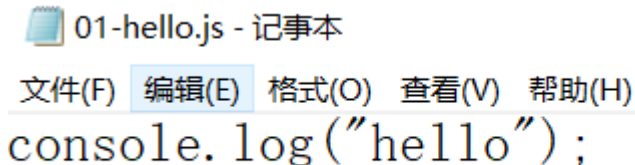
2.1、方式1：交互模式(repl模式) 仅作了解

win键+R键打开命令行窗口，输入node，即进入交互模式。在这个模式下可以输入js代码直接回车执行

```
C:\Users\VULCAN>node
Welcome to Node.js v12.13.1.
Type ".help" for more information.
> var a=5
undefined
> a
5
>
```

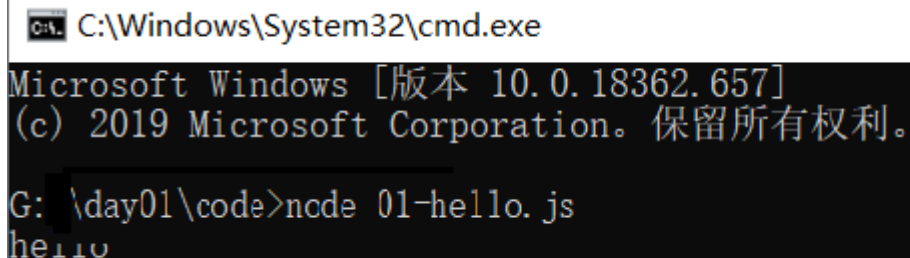
2.2、方式2：解释js文件

书写一个hello.js文件



```
01-hello.js - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
console.log("hello");
```

在黑窗口中输入node 01-hello.js回车如下图。



```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation。保留所有权利。

G: \day01\code>node 01-hello.js
hello
```

这种方式的含义是：利用node来解释js代码。（所以说node是js代码的解释器，或者运行环境）

2.3、总结

问题：现在，javascript可以运行在哪里？？

1、浏览器

2、node环境下

三、Nvm的安装和初步使用（自学）

从上一节我们知道，NodeJS有太多的版本了，切记，并不是新版本一出现，旧的版本就不去用了。

在不同的项目开发过程中，可能需要我们在电脑中同时存在多个不同版本的Node。

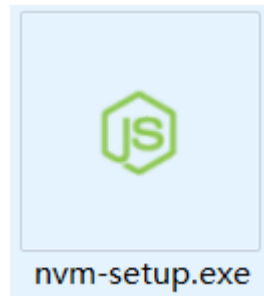
这时候就需要一个软件，来更好地管理这些不同版本地Node存在我们地电脑中，Nvm就是这样一个软件

nvm（node.js version manager 的简写）翻译过来 nodejs 版本管理器。

C:\Users\VULCAN\AppData\Roaming\nvm

3.1、安装Nvm

nvm下载链接 <https://github.com/coreybutler/nvm-windows/releases>



!!! 注意：如果电脑之前安装过nodejs，请先卸载nodejs后再进行安装。

先卸载nodejs:

控制面板 => 程序和功能 => 找到Node.js这一项，右键卸载。

再手动打开C:\Program Files目录，看看nodejs文件夹是不是真的被删除了。如果nodejs还存在，就手动删除掉。

安装nvm:

双击nvm-setup.exe文件，开始安装。

指定 nvm 安装目录，保持默认目录不变。

指定 nodejs 安装目录，保持默认目录不变。

一直下一步，直到安装成功。

检测nvm是不是安装成功了:

成功安装后，新开一个 cmd 窗口，输入 `nvm -v` 如果看到下面的信息，代表安装成功！

```
命令提示符
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\VULCAN>nvm -v

Running version 1.1.7.

Usage:

  nvm arch                : Show if node is running in 32 or 64 bit mode.
  nvm install <version> [arch] : The version can be a node.js version or "latest" for the latest stable
                                Optionally specify whether to install the 32 or 64 bit version (default
                                ).
                                Set [arch] to "all" to install 32 AND 64 bit versions.
                                Add --insecure to the end of this command to bypass SSL validation of t
                                ad server.
  nvm list [available]      : List the node.js installations. Type "available" at the end to see what
                                d. Aliased as ls.
  nvm on                    : Enable node.js version management.
  nvm off                  : Disable node.js version management.
  nvm proxy [url]          : Set a proxy to use for downloads. Leave [url] blank to see the current
                                Set [url] to "none" to remove the proxy.
  nvm node_mirror [url]    : Set the node mirror. Defaults to https://nodejs.org/dist/. Leave [url]
                                ault url.
  nvm npm_mirror [url]     : Set the npm mirror. Defaults to https://github.com/npm/cli/archive/. Le
```

配置nvm:

复制下面两句话到nvm的安装目录(C:\Users\XXXX\AppData\Roaming\nvm)下的settings.txt的最后。

(目的是加快后面下载nodejs的速度)

```
node_mirror: https://npm.taobao.org/mirrors/node/  
npm_mirror: https://npm.taobao.org/mirrors/npm/
```

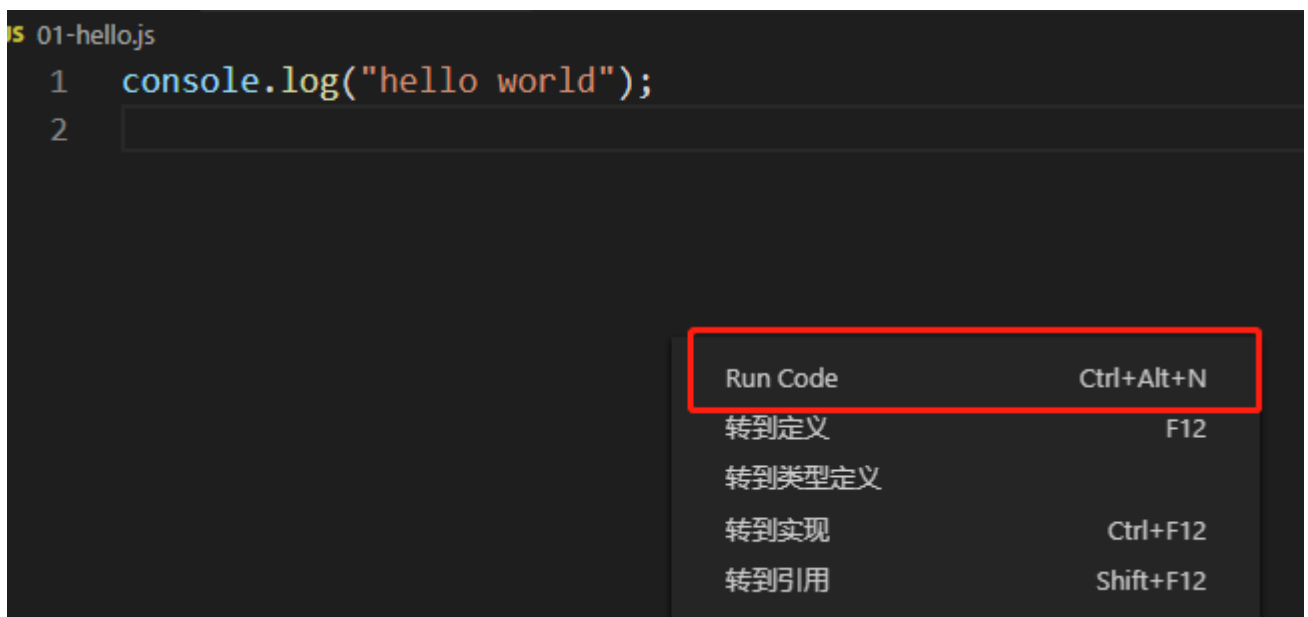
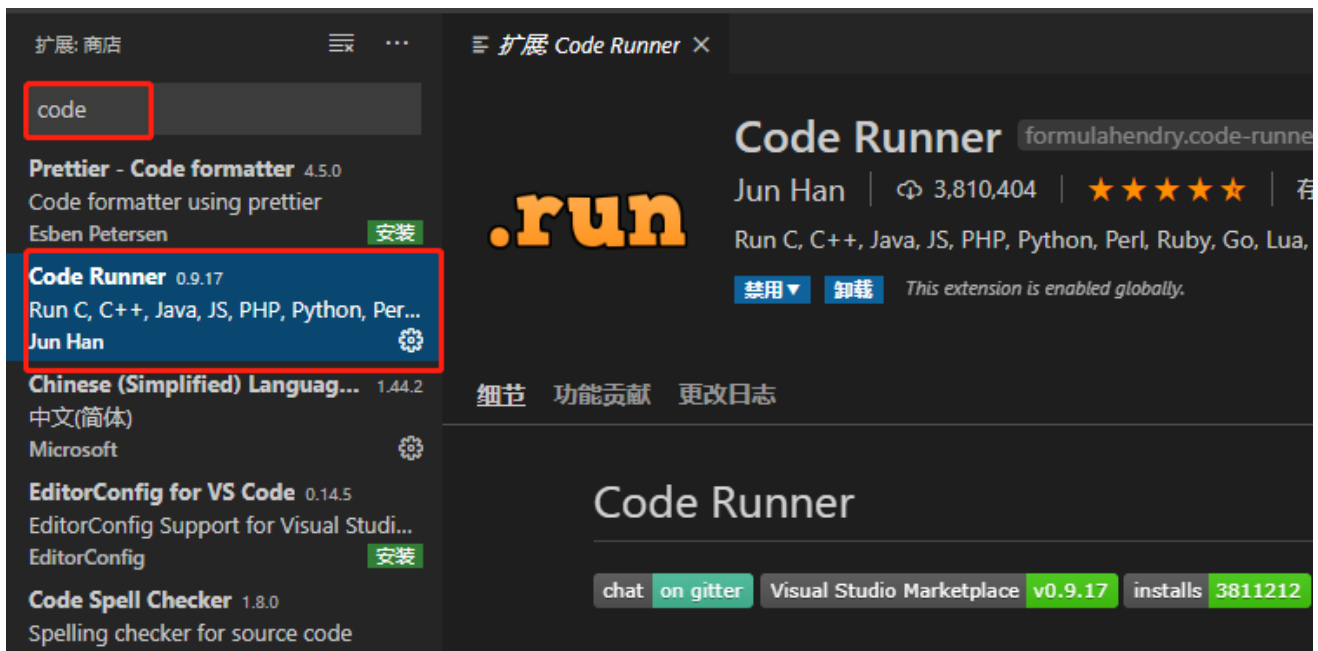
3.2、nvm命令

- nvm version : 查看 nvm 的版本
- nvm list : 查看当前安装的 Node.js 所有版本 (常用)
- nvm install 版本号 [架构] : 安装指定版本的 Node.js (常用)
- nvm uninstall 版本号 : 卸载指定版本的 Node.js
- nvm use 版本号 : 选择指定版本的 Node.js (常用)

```
# 安装指定版本  
nvm install 10.15.0  
  
# 安装最新版本  
nvm install latest  
  
# 使用安装的这个版本10.15.0  
nvm use 10.15.0  
# 查看node版本  
node -v
```

四、VScode中执行js代码

安装扩展Code Runner ,



五、Node的注意事项

1、nodejs : ECMAScript + 核心的api(重点). 没有 DOM (DOM本质上是一种接口 (API) , 是专门操作网页内容的API 标准)、BOM

```
var oDom = document.getElementById('odiv');
console.log(oDom);
var url = location.href;
console.log(url);
以上代码会报错！
```

```
[Running] node "d:\back22\d01\01-hello.js"
d:\back22\d01\01-hello.js:1
var oDom = document.getElementById('odiv');
           ^
ReferenceError: document is not defined
    at Object.<anonymous> (d:\back22\d01\01-hello.js:1:12)
    at Module._compile (internal/modules/cjs/loader.js:936:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:947:10)
    at Module.load (internal/modules/cjs/loader.js:790:32)
    at Function.Module._load (internal/modules/cjs/loader.js:703:12)
    at Function.Module.runMain (internal/modules/cjs/loader.js:999:10)
    at internal/main/run_main_module.js:17:11
```

2、nodejs 提供了文件操作系统(fs)，还提供了 web 服务的功能(http)，即使用nodejs可以编写一个web服务器（这两点在这先知道即可，后面我们会详细讲到）

六、ES6语法的简介

浏览器：javascript 三个部分：ECMAScript + BOM + DOM 服务器：javascript = ECMAScript + 系统内置的核心模块(fs http)

ECMAScript 是什么？是一个规范。ECMA 是一个组织协会，协会专门负责制定各种规则。他们制定了一个 ECMAScript 规范，规定脚本语言实现。变量声明 var function fnName 有哪些语言实现这个规范：1. JavaScript 2. actionScript (flash 动画 flash小游戏)

发展过程中 js 有很多的版本的迭代 我们前面学习的版本 ECMAScript5 版本

现在学习 ECMAScript6版本，也叫做es6版本。引入了很多新的语法特性。例如使用 let 声明变量 const 声明常量。

七、var的弊端及let关键字

使用var关键字声明变量的弊端：

- 1、var声明的变量有预解析，造成逻辑混乱，可以先使用，后声明
- 2、var可以重复定义同一个变量，逻辑错误，第二次应该是修改变量，而不是定义
- 3、var用在for循环条件中，造成for 循环的污染的问题
- 4、var 声明的变量没有块级作用域（ES5中的作用域：全局和局部）

```
// 1、var声明的变量有解析，造成逻辑混乱，可以先使用，后声明
// var a = 10;
// console.log(a);

// 2、var可以重复定义同一个变量，逻辑错误，第二次应该是修改变量，而不是定义
// var a = 10;
// var a = 30;
// console.log(a);
```

```
// 3、var用在for循环条件中，造成for 循环的污染的问题
// for(var i=0; i<10; i++){
//     console.log(i);
// }
// console.log("=====");
// console.log(i);

// 4、var 声明的变量没有块级作用域（ES5中的作用域：全局和局部）
// {
//     var b = 200;
// }
// console.log(b);
```

上面这些弊端，都在 ES6中的let关键字中得到解决：

```
// console.log(a);    //报错
// let a = 10;
// let a = 10;
// let a = 30; //报错
// for(let i=0; i<10; i++){
//     console.log(i);
// }
// console.log("=====");
// console.log(i);    //报错
{
    let b = 200;
}
console.log(b);    //报错
```

所以，let的特点：

- 1、let声明的变量没有预解析，不会有变量提升
- 2、let不可以重复定义同一个变量
- 3、let用在for循环条件中，不会造成for 循环的污染的问题
- 4、let声明的变量有块级作用域（ES6中的作用域：全局和局部还有块级作用域）

八、用const定义常量

- 1、const用来定义常量，修改就会报错
- 2、我们约定俗成地把常量都命名为大写
- 3、对象型常量中的属性可以修改
- 4、数组型常量中的每一项数据的引用可以修改

```
// var PI = 3.141592653;
// console.log(PI);
// PI = 3.5;    //不符合我们认知，因为π是个定值
// console.log(PI);
//-----
// const PI = 3.141592653;    //定义PI常量
// PI = 3.5;    //报错
```



```
//-----
// const OBJ = {
//     name: "nodejs",
//     age:11,
//     email:"nodejs@163.com"
// };
// // OBJ = {}; //报错
// OBJ.job = "nodejs@qq.com"; //可以修改成功
// console.log(OBJ); //{ name: 'nodejs', age: 11, email: 'nodejs@qq.com' }
//-----
const ARR = [10, 0, 30];
ARR[1] = 20;
console.log(ARR); //[ 10, 20, 30 ]
```

九、解构语法

9.1、对象解构

```
let obj = {
    name: "nodejs",
    age:11,
    email:"hgj@163.com"
};

// 取出所有属性并赋值：
// let name = obj.name;
// let age = obj.age;
// let email = obj.email;

// 现在只需要(等效于上面的写法):
// let {name, email, age} = obj; //{ }中的变量名和obj的属性名一致 完全解构

// 部分解构
// let {name} = obj; // 部分解构

//解构之后重命名
let {name:itsName} = obj; 解构之后重命名为itsName
```

9.2、数组解构

```
let arr1 = [10, 20, 30];

let [a, b, c] = arr1;

console.log(a); //10
console.log(b); //20
console.log(c); //30
```

```
// 部分解构
let [d] = arr1;
console.log(d); //10

let [, f] = arr1;
console.log(f); //30

// 复合解构
let arr2 = [1, 2, [10, 20, 30]];
let [j, k, [x, y, z]] = arr2;
console.log(j); //1
console.log(k); //2
console.log(x); //10
console.log(y); //20
console.log(z); //30
```

9.3、字符串解构

```
let str = "xyz";
let [a,b,c] = str;
console.log(a); //x
console.log(b); //y
console.log(c); //z
str[1] = "Y";
console.log(str); // xyz 无法修改
console.log(str[1]); // y
```

十、模板字符串(`\${变量}`)

```
var userInfo = {
  id: 1,
  name: 'andy',
  email: 'zs@163.com'
}
// 需求：希望把对象拼接为字符串："这个人的id号是1号，他的名字叫张山，email是zs@163.com";

var es5tpl = 'the userId is xxx, my name is xxx, my email is xxx';
// es6 提供一个语法表示一个字符串，使用的标识符是 反引号；
// 以前定义字符串使用的： '' ""
// 1. 允许换行 2. 允许在模板字符串里面直接使用变量

console.log(tmp1);
```

十一、es6对象的简化写法

```
/*var name = 'andy';
var age = 12;
var userInfo = {
  id: 1,
```

```

    name: name,
    age: age
  }
  console.log(userInfo);*/
  let id = 12;
  let name = 'andy';
  let age = 12;
  let fn = function(){
    console.log(`my name is ${this.name}` );
  }
  // es6 写法
  let userInfo = {
    id,
    // 如果我们的变量的名称和属性的名称一致，则可以直接只写变量名
    name, // name: name
    age,
    fn // 如果我们的函数的名称和属性的名称一致，直接写函数名即可
  }
  console.log(userInfo);
  userInfo.fn();

```

十二、ES6中函数形参的默认值

```

// ES6 之前，不能直接为函数的参数指定默认值，只能采用变通的方法
function func(x){
  x = x || 1;
  console.log(x);
}
func();

```

```

// ES6 做法，代码简洁易阅读
function func(x = 1){ // 注意当 x 为 undefined 时 x 赋值为 1
  console.log(x);
}
func();

function fun(name = 'nodejs', age = 12){
  console.log(name, age);
}
fun(); // 'nodejs', 12
fun("xiaoming", 15); // "xiaoming", 15

```

十三、函数参数的解构赋值

```

function fun({name, age}){           //{name, age} = obj      解构
  console.log(name, age);
}

let obj = {
  name: "nodejs",

```

```
    age: "11",
    email: "nodejs@163.com",
  };
func(obj);    // 输出结果： nodejs 11

// 注意：
// func(); //相当于传了一个null {name, age}=null 就会报错
// func({}); //不会报错，输出：undefined undefined
```

十四、解构赋值指定参数的默认值

```
function fun({name, age} = {}){    //防止不传实参时候的报错
  console.log(name, age);
}
fun();    //undefined undefined
```

```
function func2({name="nodejs", age=11} = {}){    //指定默认值
  console.log(name, age);
}
func2();    //nodejs 11
```

十五、rest 参数/剩余参数

用于获取函数的多余参数，这样就不需要使用 arguments 对象了。rest 参数搭配的变量是一个数组，该变量将多余的参数放入数组中。注意，rest 参数之后不能再有其他参数（即只能是最后一个参数），否则会报错。

```
function fun( a, b ,...rest){    // 把剩余的参数都交给rest
  console.log(rest);
}

fun(10, 20, 30, 50, 60);

function func2(...rest){    //rest 接收所有参数作为一个数组
  console.log(rest);
}
func2(60, 70, 80, 90);

//注意： ...符号在函数形参中使用时，则是剩余参数，其他地方不这么叫（后面说）
```

十六、扩展运算符

```
// 1、基本理解
let arr = [10, 20, 30];

function fn(a, b, c){
  console.log(a,b,c)
}
```

```

fn(...arr); //等效于：func(10,20,30);    输出结果10 20 30

// 2、合并数组
let arr2 = [40, 50, 60];
let newArr = [...arr1,...arr2]; // 等效于 [ 10, 20, 30, 40, 50, 60 ]
console.log(newArr);    //[ 10, 20, 30, 40, 50, 60 ]

// 3、合并对象
let obj1 = {
  name:"nodejs",
  age:"11",
};
let obj2 = {
  email:"nodejs@163.com",
};
let newObj = {...obj1,...obj2}; // 等效于{ name: 'nodejs', age: '11', email: 'nodejs@163.com' }
console.log(newObj);    //{ name: 'nodejs', age: '11', email: 'nodejs@163.com' }

// 4、es6中另一个合并对象的方法
let newObj2 = Object.assign({},obj1,obj2); // 把第二个及第二个以上的参数都合并到第1个上面去。
console.log(newObj2);    //{ name: 'nodejs', age: '11', email: 'nodejs@163.com' }

```

十七、箭头函数（重点）

ES6 允许使用“箭头”（=>）简化函数的定义。

```

// function func(){
//     console.log("hello");
// }

// 以上代码使用箭头函数书写为：
var func = () => {
  console.log("hello");
};
func();

```

```

// 注意点：
// 1、形参个数如果为1个，可以省略小括号不写；
// 2、如果函数体里面只有一个return 语句，可以省略大括号不写，并且他会默认返回 => 符号后面的数据。
// 3、如果函数体有多个语句，则不能省略大括号。
// 4、如果函数体只有一个语句，且返回一个对象，建议是，不要写简写的方式。

```

例子：

```

// 无参数无返回
let fun1 = () => console.log(123);
fun1();

// 无参数有返回
let fun2 = () => '888';
console.log(fun2());//undefined

```

```
// 有参数无返回
let fn3 = x => console.log('你好', x);
fn3(2);

// 有参数有返回
let fun4 = (x, y) => {
    let sum = x + y;
    return sum + '太好了';
};
console.log(fun4(1, 2));
```

注意：如果return的是单一个对象，则需要加上大括号和return，例如：

```
// let fun5 = x => {a:10, b:20};    //报错
let fun6 = x => {
    return {a:10, b:20}
};
```

十八、箭头函数中的this

箭头函数没有自己的作用域，即箭头函数 this 指向其外层作用域

```
function Person(name,age){
    this.name=name;
    this.age=age;
    this.say = function () {
        // console.log(this.name);
        setTimeout(function () {
            console.log(this.name);
        },1000);
    }
}
var p1 = new Person("zhangsan", 15);
p1.say(); // undefined

function Person2(name,age){
    this.name=name;
    this.age=age;
    this.say = function(){
        // console.log(this.name);
        setTimeout(()=>{
            console.log(this.name);
        },1000);
    }
}
var p2 = new Person2("lisi", 15);
p2.say(); //lisi

//但如果创建的是字面量对象，不适合书写箭头函数
let obj = {
    name:"nodejs",
```

```

    age: "11",
    say: () => {
        console.log(this.name);    //this不指向obj
    };
obj.say();    // undefined

```

在dom操作中使用箭头函数

```

<style>
    .box{
        width: 200px;
        height: 200px;
        background-color: pink;
    }
</style>
</head>
<body>
    <div id="odiv" class="box"></div>
    <script>
        var obj = document.getElementById("odiv");
        obj.onclick = function(){
            // setTimeout(function(){
            //     this.style.width="300px"    //修改不了
            // },1000);
            setTimeout(()=>{
                this.style.width="300px"    //修改成功
            },1000);
        }
    </script>

```

十九、面向对象(重点)

一般面向对象的语言里面，都有类，实例对象这些概念。我们通过实例化类，得到实例对象。

类：定义一个类，需要使用 class 关键字，但是es5里面，使用构造函数（首字母大写）来实现的。

继承：一般使用 extends 关键字来进行继承，但是在es5里面，也是没有extends继承，而是使用的prototype原型继承完成的。

ES6之前的写法：

```

// 用函数定义类
function Person(uname){
    this.uname = uname;
}
// 给类的实例定义方法
Person.prototype.showName = function(){
    console.log(this.name);
}
// 给类定义静态方法
Person.eat = function(){
    console.log('进食');
}

```

```

}
// 根据函数实例化对象
var p = new Person("Tom");
// 调用对象的方法
p.showName();
// 调用类的静态方法
Person.eat();

```

ES6的写法：（掌握）

```

class Person{
  // 定义构造函数
  constructor(name){
    // console.log("constructor");
    this.name = name
  }
  // 定义一个方法
  showName(){
    console.log(this.name);
  }
  //定义静态方法
  static eat(){
    console.log("吃肉");
  }
}
let obj = new Person("小张");
obj.showName();
Person.eat();

```

二十、关于继承

ES6之前的写法（仅做了解）

```

function Animal(name){
  this.name = name;
}
Animal.prototype.showName = function(){
  console.log(this.name);
}
Animal.eat = function(){
  console.log('进食');
}
// 定义子类
function Mouse(name, color){
  // 子类继承父类的属性 需要将 this 指向父类中的 name
  Animal.call(this, name);
  this.color = color;
}
// 子类继承父类的方法
Mouse.prototype = new Animal();
// 给子类实例添加新方法
Mouse.prototype.showInfo = function(){

```



```

        console.log(this.name, this.color);
    }
    var m = new Mouse('Jerry', 'gray');
    m.showName();
    m.showInfo();
    Animal.eat();

```

ES6的写法：

```

class Animal{
    // 定义构造函数
    constructor(name){
        // console.log("constructor");
        this.name = name;
    }
    // 定义一个方法
    showName(){
        console.log(this.name);
    }
    //定义静态方法
    static eat(){
        console.log("吃肉");
    }
}
class Cat extends Animal{
}
let cat1 = new Cat("小张");
cat1.showName();
Cat.eat();

```

继承中的子类constructor的写法：

在ES6中使用继承时，constructor中必须调用super()方法,其本质是在调用父类的constructor方法。通过这样的方式来达到属性继承的效果

```

class Animal{
    // 定义构造函数
    constructor(name){
        console.log("constructor");
        this.name = name;
        this.age = 0;
        this.color = "white";
    }
    // 定义一个方法
    showName(){
        console.log(this.name);
    }
    //定义静态方法
    static eat(){
        console.log("吃---");
    }
}

```

```

class Cat extends Animal{
    //注意：在ES6中使用继承时，constructor中必须调用super()方法，其本质是在调用父类的constructor方法。
    通过这样的方式来达到属性继承的效果
    constructor(name){
        super(name);
        // console.log("constructor");
        this.color = "yellow";
    }
    //定义Cat类特有的方法
    catchMouse(){
        console.log("抓老鼠");
    }
    //重写父类方法
    showName(){
        console.log(this.name, "喵喵");
    }
}

let cat1 = new Cat("Tom");
console.log(cat1.color);
console.log(cat1.age);
console.log(cat1.name);
cat1.catchMouse();
cat1.showName();

```

二十一、静态属性和静态方法

Math.max(10, 20, 30, 16) 把构造函数当成普通对象来调用某个方法，这个方法叫静态方法，如果是属性，我们就称它为静态属性。

定义静态方法，再类中通过加static关键字来定义。

关于静态属性，目前暂时没有关键字来定义，想要实现的话就在定义完类之后直接在类上添加属性，获取的时候通过类名来获取这个属性。

```

class Animal{
    // 定义构造函数
    constructor(name){
        // console.log("constructor");
        this.name = name
    }
    // 定义一个方法
    showName(){
        console.log(this.name);
    }
    //定义静态方法，再类中通过加static关键字来定义
    static eat(){
        console.log("吃肉");
    }
}

let obj = new Animal("小张");
obj.showName();
Animal.eat();

```

```
// 关于静态属性，目前暂时没有关键字来定义，要实现的话就在定义完类之后直接在类上添加属性，获取的时候通过类名来获取这个属性
Animal.type = "动物";
console.log(Animal.type);
```

二十二、全局对象

JavaScript 中有一个特殊的对象，称为全局对象（Global Object），它及其所有属性都可以在程序的任何地方访问，即全局变量。

在浏览器 JavaScript 中，通常 window 是全局对象，而 Node.js 中的全局对象是 global，所有全局变量（除了 global 本身以外）都是 global 对象的属性。

后面看到所有的全局变量，例如 console，setTimeout 和 process 是 global 变量的成员。我们甚至可以向全局变量添加成员，使其在任何地方都可用。

```
// 1. nodejs 里面没有 window全局对象，但是存在一个 global 全局对象。之前使用 console. setTimeout 这些全局的函数都是 global 上面的属性
// console.log(window); // 报错 ReferenceError: window is not defined
// console.log(global);
// 2. nodejs 里面声明的变量，并不会被挂载带 global 全局对象
let b = 20;
console.log(global.b); //undefined
// 3. 可以向global添加成员，使其在任何地方都可用
global.a = 10;
console.log(a); //10
// 4. 在nodejs执行js文件的过程中，里面也存在 this，但是这个 this 和 global 并不是相等。
console.log(global === this); //false
// 实际上，在 nodejs 里面的this代表的当前这个 js模块（暂且认为 this 代表当前这个js文件）
```

process对象(仅做了解)

```
console.log(process.argv); // 返回一个数组，前两个值是 node 命令所在位置，被执行 JS 文件的路径，若你执行命令时还有带有参数，依次会填充到此数组中也打印出来。（使用 nodejs 开发命令行的应用，需要获取 命令行的参数，才用得上）
console.log(process.arch); // 打印系统位数 x64
```

一、模块的使用

1.1、为什么要模块化(了解)

在计算机程序的开发过程中，随着程序代码越写越多，在一个文件里代码就会越来越长，越来越不容易维护。为了编写可维护的代码，我们把很多函数分组，分别放到不同的文件里，这样，每个文件包含的代码就相对较少，很多编程语言都采用这种组织代码的方式。（一个js文件就是一个模块）

使用模块有什么好处：

- 最大的好处是大大提高了代码的可维护性。其次，编写代码不必从零开始。当一个模块编写完毕，就可以被其他地方引用。我们在编写程序的时候，也经常引用其他模块，包括 Node.js 内置的模块和来自第三方的模块。

- 使用模块还可以避免函数名和变量名冲突。相同名字的函数和变量完全可以分别存在不同的模块中，因此，我们自己在编写模块时，不必考虑名字会与其他模块冲突。

说重点：主要的原因是方便项目的开发和维护。

1.2、模块规范的定义(了解)

1. 一个js文件就是一个模块，模块的作用域是私有的，内部定义的变量或者函数，只在当前的文件（模块）可以使用
2. 如果别人需要使用我们模块里面的东西，那么有两点要做(以CommonJS 的 Modules 规范：Node.js为例)
 1. 自己编写的模块，由于模块作用域是私有的，默认情况下，外部是没办法使用的；如果希望别人可以使用，则需要导出 `exports` 或者 `module.exports`。导出的时候，以对象的方式进行导出
 2. 别人要使用某个模块，则需要先引入该模块，使用 `require` 引入，引入后需要使用一个变量来接收导入的对象。

对书写格式和交互规则的详细描述，就是模块定义规范（Module Definition Specification）：

- AMD 规范：Require.js
- CMD 规范：Sea.js
- CommonJS 的 Modules 规范：Node.js
- ES6 模块化规范 `import ... from ...`

1.3、模块化使用

导出数据方式一：

```
exports.num = num;
exports.sum = sum;
exports.Animal = Animal;
```

导出数据方式二：

```
// 通过module.exports 等于一个对象，来导出数据
// 对象可采用es6简化对象的写法
module.exports = {
  num,
  sum,
  Animal
};
```

导入数据：

```
// 注意1：如果要使用某个模块里面的数据，则需要使用 require 关键字进行导入。
// 注意2：在导入用户自己开发的模块的时候，需要加上路径（1. 相对路径（多） 2. 绝对路径） 注意：./ 必须写上
// 注意3：模块文件的扩展名(后缀名)可以写，也可以不写
// 注意4：导出的模块一般需要使用一个变量来接收，一般把接收的量定义为常量
// 注意5：定义常量的名称和文件的名称保持一致（这个不是必须，大家都这么做）
const m1 = require("./modules/m1.js");
```

完整代码：

```
// m1.js中：
let num = 10;
function sum(a, b) {
    return a+b
}
class Animal{
    constructor(){
        this.age=0
    }
}
// 导出数据方式1：
// exports.num = num;
// exports.sum = sum;
// exports.Animal = Animal;
// 导出数据方式2：
// 通过module.exports 等于一个对象，来导出数据
// 对象可采用es6简化对象的写法
module.exports = {
    num,
    sum,
    Animal
};
```

```
//01-模块的使用.js
const m1 = require("./modules/m1.js");
console.log(m1);    //{ num: 10, sum: [Function: sum], Animal: [Function: Animal] }
console.log(m1.sum(10, 20));
const obj = new m1.Animal();    //30
console.log(obj.age);    //0
```

二、npm简介

npm 全称为 Node Package Manager，是一个基于 Node.js 的包管理器，也是整个 Node.js 社区最流行、支持的第三方模块最多的包管理器。npm的初衷：JavaScript开发人员更容易分享和重用代码。

1. nodejs = ECMAScript + 核心模块
2. 自己遵循 commonjs 规范写出模块，如果写的是功能模块（日期处理datejs，数字处理numberjs）。如果可以把这些模块分享出来，以后谁要进行相关功能开发的时候，直接拿开发好的模块使用即可，没必要自己在开发。在互联网有一个网站专门收集这样的工具包。<https://www.npmjs.cn/>。
3. 如果我们要使用这个网站里面的包，则需要使用一个功能，叫做 npm。

官网：<https://www.npmjs.cn/>

<https://www.npmjs.com/package/md5>

npm可以用来：

- 允许用户获取第三方包并使用
- 允许用户将自己编写的包或命令程序进行发布分享

npm安装：

npm 不需要单独安装。在安装 Node 的时候，会连带一起安装 npm。

执行下面的命令可以用来查看本地安装的 npm 的版本号。

```
npm -v
```

如果想升级 npm，可以这样

```
npm install npm --global
```

三、npm体验

以安装和使用md5模块为例：

1、项目目录下，执行命令 npm init，目录下会多一个package.json文件(这个文件1、记录项目相关信息，如项目名称，项目版本2、后期会记录项目中使用的第三方模块)

2、项目目录下，执行命令 npm install md5，这时候就会开始联网下载md5这个包，下载过程需要耐心等待，等待时间视网速而定。

3、看见以下代码表示下载完成：

```
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN npm@1.0.0 No description
npm WARN npm@1.0.0 No repository field.

+ md5@2.2.1
added 4 packages from 4 contributors in 2.988s
```

下载完后：本地项目目录下多了一个node_modules文件夹，我们刚才所下载的md5包及其相关依赖包都在这个文件夹里面了。以后我们开发中需要下载其他包，都会在下载在这个文件夹中。

4、下载完就可以在项目中去导入然后使用了：

```
var md5 = require('md5');
console.log(md5("12345789"));
```

运行就会得到一个加密字符串。

四、小练习

实现一个，数字转大写的功能 如：123 转 壹佰贰拾叁

在 <https://www.npmjs.com> 上搜索功能关键字

找对应可能用上的包，参考文档，进行安装，使用

五、nodemon包的使用

我们前面使用node的http模块书写过web服务器，但是每次改写一点代码都需要重启服务器，开发不是很方便。nodemon可以监听代码的改动自动更新，不需要重启服务器程序就可以看效果。

文档：<https://www.npmjs.com/package/nodemon>

下载：npm install -g nodemon

说明：-g 表示安装在全局，这种安装方式不同于前面的安装，它只需要安装一次，就能一直使用。安装的时候会 有一个专门的安装目录（安装完成会有提示安装位置，如果忘记了，可以通过npm root -g命令查看安装在哪里）

```
D:\nodejs>npm install -g nodemon
C:\dev\nodejs\nodemon -> C:\dev\nodejs\node_modules\nodemon\bin\nodemon.js  ← 全局安装的路径

> nodemon@2.0.2 postinstall C:\dev\nodejs\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.2 (node_modules\nodemon\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@2.0.2
added 105 packages from 49 contributors in 5.876s
```

安装成功，项目目录下，通过命令**nodemon 11-http模块.js**启动服务器即可。

```
D:\nodejs>nodemon 11-http模块.js
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node 11-http模块.js`
undefined
server is running at port 8080
[nodemon] restarting due to changes...
[nodemon] starting `node 11-http模块.js`
undefined
server is running at port 8080
```