NodeJS

- Node是一个js的运行环境，与浏览器一样。
- Node是一个应用程序，需要安装。
- 官网：nodejs.org 中文官网是: nodejs.cn
- 进入官网之后，会自动看到两个下载面板，左侧的是稳定版本，右侧的是测试版本。
- 严重声明：因为脱离了浏览器所以BOM没有了，DOM没有了，只有ECMAScript语法了。
    - console,setinterval,settimeout 都有,location,window,history,document,alert等对象都没有了
- nodejs的特点
    - 1、单线程
    - 2、非阻塞I/O
    - 3、事件驱动
- NodeJS的使用场景：高并发，I/O
- IO指的是内存和硬盘数据交换的过程。
- NodeJS的不适用场景：计算多
- Node的模块化
    - Node是CommonJs规范的实现之一。
        - commonJs规范与AMD，CMD等都属于模块化的规范。其中AMD，CMD都是前端模块化规范，属于异步加载模块规范。CommonJS规范是后端的模块化规范，属于同步加载模块规范。
    - 模块化规范相关。
    - 1、如何定义模块。
        - NodeJS中，一个JS文件就是一个模块。
    - 2、如何暴露模块内容
        - NodeJS有模块变量：
            - exports对象 : exports 是一个空对象 它负责向外暴露内容
            - require函数 : require是一个函数
            - module对象:module 是一个模块对象 它身上具备当前模块的相关信息 重要的属性就一个 exports 也是一个空对象它也负责向外暴露内容 此时exports和module.exports都负责向外暴露内容 以module.exports为准
            - __dirname : 当前模块的绝对路径 （不包含文件名）
            - __filename :当前模块的绝对路径 (包含文件名)
        - 推荐的暴露方式由：
            - exports.xx = xx;
            - module.exports = xx;
            - module.exports.xx = xx;

- 如何引入其他模块。
  - 分两种：
  - 1、引入核心模块：require（核心模块名称字符串）

```
var sum = require("./b");
console.log(sum(10,20))
```

  - 2、引入第三方模块：
    + 如果第三方模块位于node_modules文件夹中，则引入方式同上。
      - 如果第三方模块没有位于node_modeles文件夹中，则引入时require（相对路径）该相对路径必须以./开头。

- node_modules文件夹
  - 该文件夹的名字是固定的，用于存放所有的第三方模块。
  - 如果存放在该文件下，引入模块时，不需要书写相对路径，直接写模块名称即可。
  - 该文件夹可以存放于当前层级，上层级，上上层级，…直到根目录，必须是直接层级。

- **HTTP模块**

1、引入http模块
var http = require(“http”);
2、创建服务器对象
var server = http.createServer(function(req,res){
这个函数用于响应前端请求，
req：request前端发送的请求对象。
res：response 后端操作对象。

```
因为所有的请求都需要通过该函数进行处理，所以需要鉴别不同的请求，通过URL鉴别。
  req.url     URL字符串，只有从端口号后面开始的部分，协议，域名，端口都没有，hash没有。
请求类型。大写字符串GET POST PUT DELETE等。
req.connection.remoteAddress   IP地址
res.end("你会暴富")

})
```

3、监听到端口
server.listen(3000)

```
var http = require("http"); //引入了http模块
// console.log(http)
var server = http.createServer(function(req,res){
    // console.log(req,res)  req:request 前端发送的请求对象     res:response 后端操作的对象
    // console.log(req.url)
    // console.log(req.method)  //请求方式，是大写的字符串，GET，POST,PUT,DELETE
    // console.log(req.connection.remoteAddress)  //访问者的IP地址
    console.log(req)
    res.end("你会暴富")
})
server.listen(3000)
```

- **FS模块**

引入fs模块

readFile读取文件夹

var fs = require(“fs”)；

调用fs的方法，readFile 读取文件。readFile时一个异步代码。

fs.readFile(“./index.html”，function(err,data){

err表示在读取文件的过程中可能出现的错误，如果没有错误发生，err的值为null，如果有错误发生，err的值是对象。

data表示读取到的文件内容。前提是没有错误发生，如果有错误发生，data为undefined

})

appendFile 追加内容/文件

appendFile方法的作用是想文件追加内容，如果没有该文件，就会创建该文件。但不会创建文件夹。

})

```
var fs = require("fs");
// appendFile的作用是向已有的文件中追加内容。如果没有该文件，会先创建该文件，再追加。
fs.appendFile("./index.txt", "aefjweifhoiawehfoiawehfoiaweho", function(err) {
    console.log(err);
});
```

unlink删除文件

unlink负责异步删除文件，第一个参数是要删除的文件的路径。

var fs = require(“fs”)

fs.unlink(“path”,function(err){

})

```
1  var fs = require("fs");
2
3  fs.unlink("./index.txt", function(err) {
4      console.log(err);
5  })
```

rename修改名字

var fs = require(“fs”)

fs.rename(“path”,”修改会后的目标文件路径”,function(err){

})

```
// 第一个参数表示要更改名称的文件路径  第二个参数是修改之后的目标文件路径  第三个参数是回调函数
// fs.rename("./a.js", "./a/b.js", function(err) {
//     console.log(err);
// })


fs.rename("a", "b", function() {

})
```

readdir读取文件夹

var fs = require(“fs”);

fs.readdir(“path”,function(err,arr){

err 表示错误对象

arr 表示读取到的文件夹中的每一个成员的名称组成的数组。

```
var fs = require("fs");

// 读取文件夹  第一个参数是读取的目标路径
fs.readdir('1908', function(err, arr) {
    // err 表示错误对象
    // arr 表示读取到的文件夹中的每一个成员的名称组成的数组
    console.log(err)
    console.log(arr)
})
```

})

mkdir创建文件夹

var fs = require(“fs”)

fs.mkdir(“1998”,function(err){

```
var fs = require("fs");

fs.mkdir("1908", function(err) {
    console.log(err);
})
```

})

rmdir删除文件夹

rmdir方法只能删除空目录。

var fs = require(“fs”)

fs.remove(“1998”,function(err){

```
// 19081不是一个空文件夹 删除失败  因为rmdir方法只能删除空目录
fs.rmdir("19081", function(err) {
    console.log(err);
});
```

})

stat判定目标状态

var fs = require(“fs”);

第一个参数是目标路径字符串。

fs.stat(“path”,function(err,stat){

console.log(stat.isDirectory());

stat是一个对象，它有isDirecotry方法，返回一个布尔值，判定是否是目录。

})

```
var fs = require("fs")

// 第一个参数是目标路径字符串
fs.stat("./b.js", function(err, stat) {
    // stat是一个对象 它有isDirecotry方法 返回一个布尔值 判定是否是目录
    console.log(stat.isDirectory());
});
```

我们学习的每一个方法，都有一个对应的同步方法，方法名就是每一个方法名后边加Sync。例如：fs.readFile是异步读取，fs.readFileSync就是同步读取

删除传入的路径所有文件和文件夹代码。

```
//引入fs模块
var fs = require("fs");
function del(path){
    //读取传入的路径，arr是路径下的所有的文件夹，文件组成的数组。
    var arr = fs.readdirSync(path);
    for(var i = 0;i<arr.length;i++){
        //判定当前的状态
        var stat = fs.statSync(path + "/" + arr[i])
        if(stat.isDirectory()){
            //如果是文件夹的话，递归再调用自己
            del(path + "/" + arr[i])
        }else{
            //如果是文件的话，直接删除
            fs.unlinkSync(path + "/" + arr[i])
        }
    }
    //因为是同步方法，所以for循环结束之后，该目录下所有文件和文件夹都已经被删除，当前文件夹是空的。
    fs.rmdirSync(path)
}
```

```
module.exports = del;
```

> `req方法|属性`
> req.method 获取发送请求的方式，get/post
> req.on(“data”,halder) post请求时候接受数据
> req.on(“end”,halder)
> req.connection.remoteAddress IP地址
>
> `res方法|属性`
> res.setHeader(“content-type”,”text/plain;charset=utf-8”) 设置中文编码

模块方法

> `url`
> url.parse(urlstr,bool) 将一个url字符串转换成对象并返回,默认为false，此时url对象中的query属性是字符串，如果设置为true，将会把query变为一个对象。
>
> `querystring`
> qs.parse(str) 将query字符串转换成对象。

npm

npm是node自带的一个程序。用于管理所有的第三方模块。
常用命令

- 安装模块
  - npm install 模块名称
- 初始化package.json文件。
  - npm init
- 根据package.json文件安装模块。
  - npm install
- 查看版本
  - npm -v
- 清除缓存
  - npm cache clean –force
- 根据json配置文件下载模块
  - npm install

官网

http://www.npmjs.com

express

下载： npm install express

```
const express = require('express')
const app = express()


// 处理/的接口
app.get('/', function (req, res) {

    res.send('Hello World!')
})

// 处理/aa的接口
app.get("/aa", function(req, res) {
    // 获取get请求的数据
    console.log(req.query)
    res.send({
        error: 0,
        data: "success"
    });
});

app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

Sass

Sass是一个css预编译语言。

编译之前，不是css，编译之后才成为css。因为在书写css的时候，有很多不好的地方，最麻烦的就是权重问题，以及样式复用问题，计算等。

- 安装sass到node全局
  - npm install sass -g
- 作用：提供sass命定。
- 也就是说，安装了sass之后，就可以使用sass命令了。
- sass文件有两个后缀名：sass和scss，推荐使用scss。
- sass命令使用的方式：sass 要编译的scss文件 要生成的css文件

```
›sass index.scss index.css
```

- Sass语法
  - 变量
    - 定义
    - $a : 10px;
    - 等价于 var a = “10px”;

```
$a: 110px;
$b: 130px;
.header {
    width: $a;
    height: $b;
    .eye {
        width: $a;
        height: $b;
    }
}
.body {
    width: $a;
    height: $b;
}
```

- &
  - 在样式列表中表示当前伪类选择器。

```
.header {
    width: 100px;
    height: 100px;
    background-color: red;
    &:hover {
        background-color: blue;
    }
}
```

- 定义混合，复用同样的样式

```
@mixin hunhe {
    width: 100px;
    height: 100px;
}
.header {
    @include hunhe;
    .eye {
        @include hunhe;
    }
}
.body {
    @include hunhe;
}
```

```
.header {
  width: 100px;
  height: 100px;
}
.header .eye {
  width: 100px;
  height: 100px;
}

.body {
  width: 100px;
  height: 100px;
}
```

- 定义方法

```
@mixin rect($w, $h, $c) {
    width: $w;
    height: $h;
    background-color: $c;
}
```

```
.header {
    @include rect(100px, 200px, red);
    .eye {
        @include rect(50px, 100px, blue);
    }
}
```

```
.header {
  width: 100px;
  height: 200px;
  background-color: red;
}
.header .eye {
  width: 50px;
  height: 100px;
  background-color: blue;
}
```

- 方法参数默认值

```
@mixin method($w:100px, $h:200px, $c:orange) {
    width: $w;
    height: $h;
    background-color: $c;
}

.header {
    @include method();
}
```

```
.header {
  width: 100px;
  height: 200px;
  background-color: orange;
}
```

- if语句

```
@mixin method($w, $h, $c: pink) {
    width: $w;
    height: $h;
    @if $w > 500px {
        background-color: red;
    } @else if $w > 400px {
        background-color: blue;
    } @else if $w > 300px {
        background-color: orange;
    } @else {
        background-color: $c;
    }
}

.header {
    @include method(501px, 400px);
}
.body {
    @include method(401px, 400px)
}
```

```
.header {
  width: 501px;
  height: 400px;
  background-color: red;
}

.body {
  width: 401px;
  height: 400px;
  background-color: blue;
}
```

- for循环语句

```
// to 无法到达最后一次循环数字
// @for $i from 0 to 12 {
//     .col-lg-#{$i} {
//         width: percentage($i / 12) ;
//     }
// }
```

```
// through 可以到达最后一次的循环数字
@for $i from 0 through 12 {
    .col-lg-#{$i} {
        width: percentage($i / 12) ;
    }
}
```

```
.col-lg-0 {
  width: 0%;
}

.col-lg-1 {
  width: 8.3333333333%;
}

.col-lg-2 {
  width: 16.6666666667%;
}

.col-lg-3 {
  width: 25%;
}

.col-lg-4 {
  width: 33.3333333333%;
}
```

- while循环

```
$i: 0;
@while $i <= 12 {
    .col-lg-#{$i} {
        width: percentage($i / 12);
    }
    $i:$i + 1;
}
```

```
.col-lg-0 {
  width: 0%;
}

.col-lg-1 {
  width: 8.3333333333%;
}

.col-lg-2 {
  width: 16.6666666667%;
}

.col-lg-3 {
  width: 25%;
}

.col-lg-4 {
  width: 33.3333333333%;
}
```

- 引入sass文件

```
@import "./common.scss";

.header {
    width: $a;
    height: $a;
    background-color: $color;
}
$a: 100px;
$color: rgba(55, 66, 123, .6);
```

gulp

gulp是一个NodeJS的第三方模块。
gulp是一个工程化工具。可与将我们书写的代码编译，压缩，合并，重命名等。
gulp有许多版本，我们学习的是3.9.1版本。

- 安装gulp
  - 下载
    - npm install gulp@3.9.1 -g 安装到全局
    - npm install gulp@3.9.1 安装到本地，通常是指你要工程化操作的项目的根目录。
  - 书写配置文件：gulpfile.js
  - gulp自带的API
    - gulp.task(taskNmae,content);

        - taskName 可以是字符串
        - content 可以是数字，可以是函数
            - 如果是数组，数组中的每一个成员都必须是其他任务的名字。
            - 如果是函数，会执行函数体中的代码。
    - gulp.src(path) 用于定位文件和文件夹。
        - path 路径字符串，可以用*代替所有
    - gulp.src().pipe() 该方法用于设置如何操作选中的文件。该方法可以链式调用。pipe的参数是具体操作。
    - gulp.dest() 方法发是就·具体的操作·，可以放入pipe中，表示发布到哪里去，参数是发布的路径字符串。
        - 如果字符串锁指向的路径已经存在则使用，如果不存在先创建再使用。
    - gulp.watch() 两个参数，第一个参数可以是任务名称。可以是数组。第二个参数是要执行的任务名称组成的数组，也可以是函数。当第一个参数一发生变化，就去执行第二个参数的任务。

插件集合

gulp-autoprefixer **添加css样式前缀**

```
var gulp =require("gulp");
var autoprefixer = require("gulp-autoprefixer");

gulp.task("css-prefixer", function() {
    // 定位资源
    gulp.src("./origin/css/reset.css")
    // 定义操作
    .pipe(autoprefixer())
    // 定义操作
    .pipe(gulp.dest("./publish/css/"));
})

// 定义默认任务
gulp.task("default", ["css-prefixer"])
```

gulp-babel@7.0.1 **转换ES6为ES5**

```
// 使用gulp将ES6代码转换为ES5代码
var gulp = require("gulp");
var babel = require("gulp-babel");

gulp.task("es6toes5", function() {
    gulp.src("./origin/js/a.js")
    .pipe(babel({
        presets: ["es2015"]
        //这里使用转换成es5的功能需要依赖两个模块，得用npm安装，
        //分别是babel-core和babel-preset-es2015/babel-preset-env
    }))
    .pipe(gulp.dest("./publish/js"));
})


gulp.task('default', ["es6toes5"])
```

依赖core和preset

babel-core

babel-preset-es2015/ babel-preset-env

gulp-htmlmin **处理html**

```
var gulp = require("gulp")
var htmlmin = require("gulp-htmlmin")

gulp.task("htmlmin", function(){
    gulp.src("./origin/html/index.html")
    .pipe(htmlmin({
        collapseWhitespace : true
        //把所有的空格都去掉
    }))
    .pipe(gulp.dest("./pulish/html"))
});
gulp.task("default",["htmlmin"])
```

gulp-sass **处理sass任务**

```
var gulp = require('gulp')
var sass = require("gulp-sass")

gulp.task("sass",function(){
    gulp.src("./origin/scss/*.scss")
    .pipe(sass())
    .pipe(gulp.dest("publish/css"))
});
gulp.task("default",["sass"])
```

gulp-clean **清除文件**

```
var gulp = require("gulp")
var clean = require("gulp-clean")

gulp.task("clean",function(){
    gulp.src("./publish/**")
    .pipe(clean())
})

gulp.task("default",['clean'])
```

gulp-sequence **任务顺序管理**

```
var gulp = require("gulp")
var sass = require('gulp-sass')
var clean = require("gulp-clean")
var sequence = require("gulp-sequence")
gulp.task("clean", function(){
    console.log(4)
    return gulp.src("./publish/**")  //这里必须return
    .pipe(clean())
})
gulp.task("sass", function(){
    console.log(3)
    return gulp.src("origin/scss/*.scss")
    .pipe(sass())
    .pipe(gulp.dest("publish/scss/"))
})

gulp.task("default", sequence("clean", "sass"))
```

gulp-webserver **启动自动打开文件,创建服务器**

```
var gulp = require("gulp")
var webserver = require("gulp-webserver")
var sequence = require("gulp-sequence")
gulp.task("webserver", function(){
    return gulp.src("./publish/")
    .pipe(webserver({
        host : "localhost",
        port : 3000,
        livereload : true,
        open : "./index.html",          //代码正常执行打开的网页。
        fallback : "html/index.html",   //出错时候打开的网页
        proxies : [
            {
                source : "/a.php",
                target : "http://localhost:80/a.php"
            },
            {
                source : "/b/",
                target : "http://localhost:80/b/"
            }
        ]
    }))
})
gulp.task("publish", function(){
    return gulp.src("./origin/**")
    .pipe(gulp.dest("./publish/"))
})
gulp.task("default", sequence("publish","webserver"))
```

gulp-uglify **压缩js文件**

```
gulp.task("uglify", function() {
    gulp.src('./origin/**/*.js')
    .pipe(uglify())
    .pipe(gulp.dest("./publish/**/"))
})
```

gulp-cssmin **压缩css文件**

```
gulp.task("cssmin", function() {
    gulp.src('./origin/**/*.css')
    .pipe(autoprefixer())
    .pipe(cssmin())
    .pipe(gulp.dest("./publish/**/*.css"))

})
```

NVM

npm是Node的模块管理工具

NVM是一个Node的版本管理工具
因为Node有很多的版本，有一些高版本与低版本之间有差异，此时就需要将node的版本切回低版本。

> 命令
> nvm ls-remote 查看有哪些node版本可以安装
> nvm 用于管理node 版本
> nvm list 查看当前所有的node 版本
> nvm install v10.13.0 安装指定的版本，安装多版本
> nvm use —delete-prefix 10.13.0 使用nvm use切换到指定的版本
> nvm current 查看当前node版本
> nvm alias default 命令来指定一个默认的node版本

git

git是一个分布式版本控制器
是管理文件的一个工具。

- **git 的三个区域**
  - 工作区
    - 该目录下就是你的工作区，可以创建文件和文件夹等。
  - 暂存区
    - 暂时存储区
  - 历史记录
    - 真正的存储区域
- **指令**
  - git init 负责将当前目录纳入git管理，表示初始化。会形成 .git 文件夹，该文件夹千万不要动它。注意，初始化后的文件夹是隐藏的。
  - git add 文件\路径. 负责将文件或者文件夹添加到暂存区。真正的存储区域。当文件修改之后需要重新添加到暂存区。
    - git add js/ 将js文件夹下所有的文件加入工作区
    - git add . 将当前目录所有文件加入工作区
  - git status 查看当前的工作区状态
  - git commit -m “描述信息” 负责将暂存区的内容形成一个历史记录
  - git log 查看历史操作记录
  - git reset –hard 历史记录id 注意历史记录id是一个参数，该指令用于切换到指定的历史记录状态。
- **git分支**
  - git 默认的分支名称为master

- 分支是指对整体的一个克隆
- git默认有一个主分支
  - git branch 分支名称 以当前所在的分支为模板克隆一个新分支
  - git checkout 分支名称 切换到新分支
  - git checkout -b 分支名称 创建并切换到新分支