目录

- `nodeJS`
- `nvm` node管理工具
- `gulp`
- `js` 模块化历程（`require.js` `AMD` 规范 异步执行）

1.Gulp的准备工作 (和postcss一样,都是第三方包)

- `gulp` 将开发流程中让人痛苦或耗时的任务自动化，从而减少你所浪费的时间、创造更大价值。(自动整理开发文件夹)

1. 下载NodeJS
2. 启动NodeJS
   - `windows`： `windows` 键 => `nodejs` 文件夹 => `nodejs prompt`
   - `node -v` 查看安装 `nodeJS` 版本
   - `nvm` 自行查阅
3. 安装NodeJS
   `npm` 管理第三方包的管理器 (下载源是国外的网站)
4. 安装 `cnpm` 淘宝镜像：https://npmmirror.com/
   - 在 `nodejs` 窗口安装 `cnpm`
   - `npm install -g cnpm --registry=https://registry.npmmirror.com`
5. `cnpm` 下载 是从国内淘宝服务器上下载的数据,版本可能会落后一点
   - 删除插件： `cnmp uninstall 插件名`
   - 删除全局模块插件： `npm uninstall -g xxx`

6. 安装 `Gulp` 到本地
   1. 全局安装 `Gulp`
      `cnpm install gulp -g`
      `gulp -v` 查看安装 `gulp` 的版本号
      - 苹果电脑电脑的每一条命令前面都必须加 `sudo` (根权限，可能需要输入密码)
   2. 进入到你想开发项目的目录
      - DOS命令:

描述	命令
切换磁盘	`c:` `d:`
显示当前目录下的目录信息	`dir`
改变目录	`cd` 目标目录

描述	命令
回退	`cd..` 回退根目录: `cd/`
清除所有命令行并返回根目录	`cls`
补全	`Tab` 键
创建目录	`md 目录`
删除目录	`rd 目录`
复制内容	`copy 文件名 目标目录:\文件名`
删除文件	`del 文件名/目录名` (删除里面的文件)

7. 初始化当前目录

   `cnpm init`

   - 会让你依次填写以下信息

信息	描述
package name	(cnpm) 包名
version	版本
description	描述
entry point	(index.js) 入口文件
test command	关键字(可被人在网上查找到)
git repository	仓库
keywords	关键词
author	作者
license	(ISC) 许可证、协议

     - 生成 `package.json` 的文件, 存放当前项目一些配置信息
       - 不能在 `json` 文件中添加注释,安装和修改会出错

8. 在当前项目里面安装 `gulp` 到本地

   `cnpm install gulp@3.9.1 --save-dev`

   - 简化命令: `cnpm i gulp@3.9.1 -D`
     - `--save` 将这个文件, 安装到本地的当前文件夹
     - `-dev` 将安装 `gulp` 的信息保存在 `package.json` 里面

        - `@` 是为了指定版本号, 新版和旧版语法不通
    - 安装完成的`gulp`版本是`4.0.2` 和`4.0`之前的语法不一致
        1. 将`package.json`中的版本号删除
        2. 将`package.json`同级下的`node_modules`文件夹删除
        3. 重新下载： `cnpm install gulp@3.9.1 --save-dev`

9. 在文件夹创建一个`gulpfile.js`的文件, 专门`gulp`去编写任务的
    - 如果报错`ReferenceError: primordials is not defined`， 是因为`node`版本和`gulp`版本不兼容,将`node`降低版本-> `12.xx`
        - 如果没有将`gulp`路径添加到`path`路径, 只能在`gulp`文件夹内`cmd`

    - 扩展: `nvm`管理多个`node`版本
        1. 安装nvm： https://blog.csdn.net/weixin_44582077/article/details/110237056
        2. 将`nvm`文件夹路径加入path路径变量： `C:\Users\baole wang\AppData\Roaming\nvm`
        3. nvm命令

命令	描述
`nvm off`	禁用`node.js`版本管理(不卸载任何东西)
`nvm on`	启用`node.js`版本管理
`nvm install <version>`	安装`node.js`的命名 `version`是版本号 例如： `nvm install 8.12.0`
`nvm uninstall <version>`	卸载`node.js`是的命令，卸载指定版本的nodejs， 当安装失败时卸载使用
`nvm ls`	显示所有安装的`node.js`版本
`nvm list available`	显示可以安装的所有`node.js`的版本
`nvm use <version>`	切换到使用指定的nodejs版本
`nvm v`	显示`nvm`版本

10. `gulpfile.js`中编写任务,需要在控制台 通过
    - `gulp 任务名` ： 运行你编写好的程序

```
/*  commonJS规范：1. require() 将模块引入；  2. 使用这个模块上的函数 */
const gulp = require("gulp");
// 编写第一个任务
  // 第一个参数 ： 任务的名字, 自定义
  // 第二个参数 ： 回调函数, 任务执行的功能(可以传入数组)
gulp.task("hello",function(){
    console.log("hello world");
})
```

- 使用: 进入此目录 `gulp hello`
- 在 `DOS` 窗口运行的js脚本,只能编写 `ECMAScript`,即 `ECMA-262`
  - 一个完整的 `JavaScript` 由 `ECMAScript` 、 `DOM` 、 `BOM` 组成

11. Gulp的基本函数
    - `gulp.task('任务名', [函数1,函数2])`： 创建任务
    - `gulp.src( [globs] )` 找到源文件路径
      - `*` 此目录下的 文件,不包括文件夹
      - `**` 此目录下的 文件、文件夹、文件夹内的文件
    - `gulp.dest(目标路径)` 找到目的文件的路径
      - 目的文件路径不存在会自动创建
    - `pipe()` 程序运行的管道

    1. 整理 `.html` 文件

```
gulp.task("copy-html",function(){
  return gulp.src("index.html")
  .pipe(gulp.dest("dist/"))
})
```

    2. 拷贝多级目录图片

```
gulp.task("image",function(){
  // 拷贝 img内部 文件夹内的图片
  return gulp.src("img/*/*").pipe(gulp.dest("dist/image"))

  // 拷贝 img内的图片 和 img内部 文件夹内的图片
  return gulp.src("img/**/*").pipe(gulp.dest("dist/image"))
})
```

       - `{}` 大括号内可以写多个后缀名,用 `,` 隔开

```
gulp.task("image",function(){
  return gulp.src("img/*.{jpg,png}")
  .pipe(gulp.dest("dist/image"))
})
```

    3. 拷贝多个后缀不同的文件 到一个目录中

```
gulp.task("data",function(){
  return gulp.src(['xml/*.xml','jsp/*.jsp','!xml/04.xml'])
  .pipe(gulp.dest("dist/data"))
})
```

    4. 一次执行多个任务

```
gulp.task('build', ['copy-html','image', 'data'],function(){
  console.log('任务执行完毕')
})
```

- `'!路径'` 排除这个路径

```
gulp.task('任务名',function(){
  return gulp.src(['路径1','路径2','!路径3','img/*.{后缀1,后缀2}'])
  .pipe(gulp.dest('目标路径'))
})
```

2.gulp监听

- `postcss` 监听：`postcss src/demo.css -o dist/demo.css -w`
  - `-w` 就是监听
- `gulp` 启动监听 `galp.watch( '要监听的路径',['任务1', '任务2'] )`

```
gulp.task('watch',function(){
  gulp.watch('index.html',['copy-html']);
  gulp.watch('img/**/*',['image']);
  gulp.watch(['xml/*.xml','jsp/*.jsp','!xml/04.xml'],['data']);
})
```

  - 第一个参数: 要监听的文件
  - 第二个参数: 要执行的任务（数组）

3.gulp插件

- 地址: https://gulpjs.com/plugins/
- 使用第三方插件的步骤:
  1. 将插件下载到本地
     - `cnpm install 插件名字 --sava-dev`
     - 简写: `cnpm i 插件名字 -D`
  2. 通过 `require()` 引入插件
  3. 查阅插件用法 使用插件

- 处理css的插件
  - `gulp-sass()` ： 编译scss插件 (和vscode插件无区别)

```
var sass = require('gulp-sass');
gulp.task('sass',function(){
  return gulp.src('stylesheet/index.scss')
  .pipe(sass())
  .pipe(gulp.dest('dist/css/index2.css'))
})
```

  - `gulp-minify-css()`： 压缩css

```
const minifycss = require('gulp-minify-css')
gulp.task('minifycss',function(){
  return gulp.src('stylesheet/index.css')
  .pipe(minifycss())
  .pipe(dest('dist/css/minifycss'))
})
```

    - 我使用的是`gulp-cssnano()`：进行`css`压缩

```
var cssnano = require('gulp-cssnano')
gulp.task('cssnano',function(){
  return gulp.src('stylesheet/index.css')
  .pipe(cssnano())
  .pipe(dest('dist/css'))
})
```

  - `gulp-rename(新文件名)`：重命名
    - `cssnano`和`rename`组合 (同时生成未压缩 和 压缩`css`)

```
const cssnano = require('gulp-cssnano')
const rename = require('gulp-rename')
gulp.task('rename',function(){
  return gulp.src('stylesheet/index.css')
  .pipe(dest('dist/css')) // 生成未压缩文件

  .pipe(cssnano())    // 压缩css
  .pipe(rename('index2.min.css')) //对css重命名
  .pipe(dest('dist/css'))     //生成文件
})
```

- 处理js插件
  - `gulp-concat(新文件名)`：合并文件

```
const concat = require('gulp-concat')
gulp.task('concat',function(){
  return gulp.src('javascript/*.js')
  .pipe(concat('all.js'))
  .pipe(dest('dist/js'))
})
```

  - `gulp-uglify`：压缩`js`文件

```
const uglify = require('gulp-uglify')
gulp.task('uglify',function(){
  return gulp.src('javascript/*.js')
  .pipe(uglify())
  .pipe(dest('dist/js'))
})
```

- `gulp-connect`：启动一个本地服务器

```
const connect = require('gulp-connect');
gulp.task('connect',function(){
  connect.server({
    root:"dist",      // 服务器的根目录
    port:"8888",
    livereload: true,    // 启动实时刷新功能
  })
})
```

- `livereload: true` : 启动实时刷新功能
  1. 在`connect.server` 中 添加 `livereload : true`
  2. 编写 `gulp.watch` 中 编写 `gulp.watch('监听路径',['任务1','2']);`
  3. 在 任务 函数中的末尾 添加 `connect.reload()`

     ```
     gulp.task("copy-html",function(){
       return gulp.src("index.html").pipe(gulp.dest("dist/"))
       .pipe(connect.reload())
     })
     ```

  4. 同时启动服务器和监听

     ```
     gulp.task('default',['watch','connect'])
     ```

     - `default` 可以直接在控制台 通过 `gulp` 直接启动
  - 只要修改 监听路径的文件, 就会刷新服务器
    - 不能修改`dist`(gulp任务生成)中的代码

- 在`package.json`中 会显示 项目中用到的所有插件
  - 当项目拷贝给别人的时候, 把`node_modules`文件夹删除, 此文件夹存放的是插件
  - 当项目从别人那边拿过来的时候, 在文件夹安装 `gulp`
    1. 安装 `nvm` 安装 `node@8.12.0`
    2. 安装`cnpm`
    3. 在文件夹安装`gulp`
    4. 启动 `gulp`全局监听

4.javascript模块化 (ECMA6之前 的 写法,ECMA6不用这么写)

- `javascript`不支持 类 和 模块, 但`ECMA6`正在制定 类 和 模块, 但还需要很长时间才能投入使用
- js 模块发展的历史
  1. 原始写法 -> 函数封装: `function showA(){ console.log('a') }`
     - 问题: 全局变量污染
  2. 对象写法 -> 对象中写 变量 函数: `var moudleB = { showA: function(){ } }`
     - 解决 全局变量污染
     - 问题 变量暴露
  3. 立即执行函数闭包

- 解决 全局变量污染、变量暴露
- 问题 无法扩展

4. 放大模式 -> 将已经闭包的对象 传入新的闭包,实现扩展
   - 解决 全局变量污染、变量暴露、无法扩展
   - 问题 `js` 的 `src` 引入是异步, 无法保证顺序

```
moudleA = (function(mod){
  var showC = function(){
      alert("hello world")
  }
  mod.showC = showC
  return mod
})(moudleA)
```

5. 宽放大模式 -> 方法模式基础上修改
   - 改变方法模式中立即执行函数的形参： `(function(mod){})(moudle || {})`
   - 解决 全局变量污染、变量暴露、无法扩展、引入异步

5.模块规范

- `.js` 文件就是一个模块

1. `CommonJS` 规范 (服务器规范) 编写代码时 同步执行 -> `nodejs`
   需要先下载 模块
   - 声明:暴露

```
moudle.exports = {
  outA: showA,
  outB: showB
}
```

   - 引入:(同步执行)

```
var moudleA = require(moudle)
moudleA.outA();
moudleA.outB();
```

2. `AMD` 规范: (客户端/浏览器) 异步执行 `require.js`
   需要 下载 `require.js`
   - 声明:

```
define(function(){
  return {
    outA: showA,
    outB: showB
  }
})
```

- 引入:(异步执行)

```
require( [模块2,模块1], function(obj){})
require(['moudleA.js'],function(moudleA){
  // 这里的代码,模块引入之后执行
  moudleA.outA;
  moudleA.outB
})
alert("hello world")
```

3. 小彩蛋: `CMD` 规范 阿里 `ECMA6` (模块化规范)
   - 声明:

```
export = {
  outA: showA,
  outB: showB
}
```

   - 引入: (异步)

```
import moudleA from "moudleA.js"
moudleA.outA();
moudleA.outB();
```

6. require.js AMD规范

- 最早的时候, 所有 `javascript` 代码都写在一个文件里面,只要加载这一个文件就够了;后来代码越来越多,就必须分成多个文件,依次加载
- `require.js` 解决两个问题
  1. 实现 `js` 文件的**异步加载**,避免网页加载 `js` 失去响应
  2. 管理模块之间的依赖性, 便于代码的编写和维护
  - 官网: https://requirejs.org/
    - 将 页面代码复制到本地
- script标签

```
<script src="./js/require.js" async="true" defer data-main="js/main"></script>
```

  - `async = 'true'` : 异步加载 (IE: `defer`)
  - `data-main = ""` : 设置入口文件
    - 入口文件: 管理当前 `.html` 页面使用的所有 `js` 代码
    - 每一个 `.html` 文件都要对应一个入口文件, 而且名字不能重复
  - 如果多个 `script` 文件都添加了 `async` 属性, 那么添加 `async` 属性的文件会依次异步加载
    - 后续引入模块, 在 `data-main` 中的 `js` 中引入
- (模块.js)声明模块 遵从 `AMD` 规范

```
define(function(){
  function add(x,y){
    return x+y
  }
  function show(){
    console.log("hello world")
    return
  }
  // 对外暴露
  return {
    outAdd: add,
    outShow: show
  }
});
```

- (入口文件)引入模块 遵从 `AMD` 规范

```
require([模块1,模块2], function(obj1, obj2){})
require(["demo/add"],function(addObj){
  console.log(addObj.outAdd(10,20))
  addObj.outShow()
})
```

  - 配置路径, 引入模块的时候就不用再写路径了(入口文件配置)

```
require.config({
  paths: {
    // 在此页面显示的模块名 : 模块的路径
    add: 'demo/add'
  }
})
require(['add'],function(addObj){ })
```

  - (入口文件.js)引入多个模块

```
require.config({
  paths:{
    add : 'demo/add',
    mul : 'demo/mul'
  }
})
reruire(['add', 'mul'],function(add, mul){
  add.属性
  mul.属性
})
```

- (模块.js) 如果一个模块内部, 依赖其他模块

```
define(['模块路径','config声明引入过的模块名'],function(a1,a2){
  a1.属性
  a2.属性
})
```

- 总结:
  - 声明模块 (`模块.js`)

    ```
    define(function(){})
    ```

    - 依赖其他模块时,引入其他模块

      ```
      define( [模块路径, config内部引入的模块名], function(a1, a2){})
      ```

  - 配置路径 引入模块 (`入口.js`)

    ```
    require.config({
      paths:{
        add: 'demo/add'
        名称: 路径
      }
    })
    ```

    - `入口.js` 使用其他模块

      ```
      require([ 模块路径, config内部引入的模块名],function(add, mul){})
      ```

  - `index.html` 引入的 `js`
    - `async = 'true'` : 异步加载 (IE: `defer`)
    - `data-main = ""` : 设置入口文件
      - 每一个 `.html` 文件都要对应一个入口文件, 而且名字不能重复

- 扩展: `require.config({})` 中的 `shim`
  - 设置引入模块路径的依赖关系-> `JQcookie` 依赖 `JQ`

```
require.config({
  paths:{
    'jquery':'路径/jquery-3.6.0',
    'jqCookie': '...'
  },
  shim:{
    'jqCookie': "jquery",
    "依赖者": "被依赖者"
  }
})
```

7.模块化案例

- 使用模块化, 放大div 拖拽div
- 可以在模块中 获取 节点

8.模块化 实战项目

`node` 版本 `11.4.0`