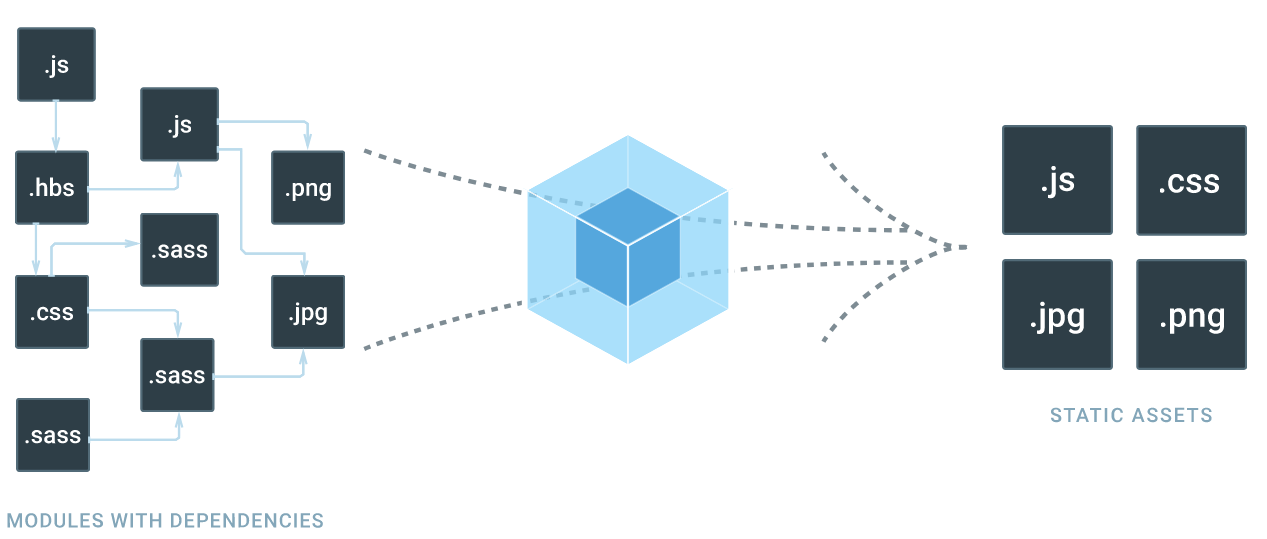
爱旅行项目—webpack使用指南

为了能满足前端的开发需求，减少不必要的人工投入，我们这里使用webpack进行文件的打包和编译，使我们更方便的进行前端代码的开发和维护。Webpack可以将多种静态资源js、css、less转换成一个静态的文件（如图1），减少页面的请求，同时也减少了我们去转义less或ES6语法等工作，大大的提高了我们的开发效率；在学习webpack之前请先看《npm使用指南》，关于npm命令此文档不做过多的讲解；

关于webpack的使用我们简单的从以下几点进行讲解：安装webpack、使用webpack打包部署、webpack配置文件、本地环境调试、webpack配置文件的编写；  
 图1

# 1.1 安装webpack

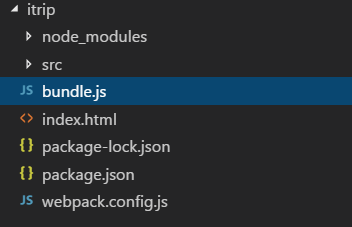
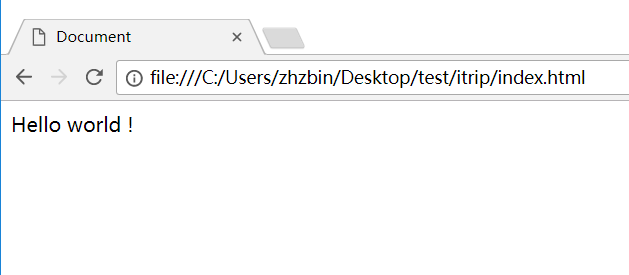
在安装webpack前需要先安装nodejs，由于npm安装比较慢我们可以使用淘宝的镜像cnpm（可参照《npm使用指南》）进行安装，这里简单介绍webpack是如何进行全局安装和项目安装：

* + 全局安装：cnpm install webpack -g
    - 安装完成之后可以打开cmd命令窗口输入webpack -v 会打印出版本号证明你已经安装成功；
  + 当前项目安装：cnpm install webpack --save-dev （此命令是针对于项目进行安装的）
    - 项目安装完成之后会在package.json文件devDependencies属性下有对应的信息"webpack": "^3.4.1+"

# 1.2 使用webpack

安装完成webpack之后我们来讲一下关于webpack的使用，主要从以下几点进行讲解：打包部署、文件加载；

为方便我们在后期进行项目分组开发，使用VSCode打开itrip文件夹，打开终端输入初始化命令npm init 进行初始化，初始化完成之后我们就可以开始安装对应的插件了。

* 一、在itrip文件夹下建src文件夹，在此文件夹下新建一个index.js文件；
* 二、在index.js文件中添加一下代码document.write(“Hello world !”)
* 三、在itrip下新建一个index.html文件，此文件的代码如下  
  <html>   
   <head>   
   <meta charset="utf-8">   
   </head>   
   <body>   
   <script type="text/javascript" src="bundle.js" charset="utf-8"></script>  
   </body>  
   </html>
* 四、在终端输入webpack ./src/index.js bundle.js命令后文件夹里会出现bundle.js文件（图2）。进入到itrip文件夹下面，用浏览器打开index.html文件（图3），这就表明我们已经打包完成了；  
    
   图2  
    
   图3
* 五、现在我们可以添加第二个js文件了代码如下  
  export let world=()=>{  
   document.write(" This is index2.js.");  
  }  
  index.js文件代码如下  
   import {world} from './index2.js'  
   world();  
  代码修改完成之后输入 webpack ./src/index.js bundle.js命令之后，再打开页面显示出我们在index2.js文件中输入的内容；
* 六、引入CSS样式进行编译，在src文件夹下新建style.css文件，在文件夹里面添加：body{background-color:red},然后在index.js文件中添加 import ‘style.css’,然后使用 webpack ./src/index.js bundle.js命令打包会报错，提示编码错误，不能解析CSS样式，我们需要对引入的信息修改成：import "!style-loader!css-loader!./style.css"，再输入打包命令，发现还是报错，提示缺少css-loader style-loader这两个文件，那我们可以使用npm install css-loader style-loader --save-dev 将文件添加到package.json文件中，如果在后期打包部署时还是报文件缺失，可使用此命令添加文件；添加完成之后重新打包部署，页面显示为红色；

# 1.3 webpack配置文件

在上面的使用过程中，我们需要对不同的文件打包的时候引入对应的解析插件，但是在整个项目的开发过程中，使用这种方式非常不利于企业级开发，而且也比较浪费时间，这里我们使用webpack.config.js文件进行部署；

我们在itrip文件夹下新建一个webpack.config.js文件，在文件里面添加一下文件信息：  
module.exports = {

entry: "./src/index.js",

output: {

path: \_\_dirname,

filename: "bundle.js"

},

module: {

loaders: [

{

test: /\.css$/,

loader: "style-loader!css-loader"

}

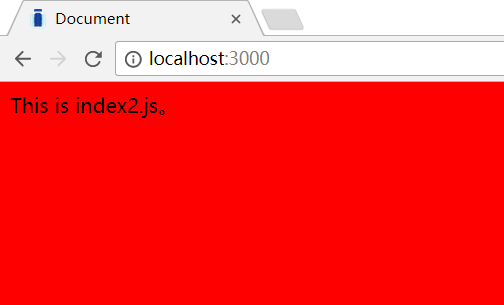
]

}

};

将index.js文件中的’ !style-loader!css-loader!’删除，直接运行webpack即可，webpack会自动查找webpack.config.js文件；关于webpack.config.js文件的其他配置属性请参照：<https://webpack.js.org/concepts/loaders/#example>地址

# 1.4 本地调试

我们在开发过程中发现，每次修改代码都要手动的build一下，页面再刷新一次才能看到后面修改的界面效果，那是否有别的方式来解决我们这个问题那？答案是有的，我们需要安装使用命令进行安装：cnpm install webpack-dev-server -g（全局安装）安装完成之后输入一下命令进行启动：webpack-dev-server --progress –colors 在浏览器打开 http://localhost:8080/ 可以访问页面，也可以通过 --port 参数修改端口，参数后面加端口名称例如：webpack-dev-server --progress –colors --port 3000 在页面访问的地址变成：http://localhost:3000/，界面显示效果如下，图4；  
  
 图4

为方便我们在后期开发时减少命令的输入我们在package.json文件中增加以下代码，再输入时可使用 ：npm start进行启动操作，打包部署使用 npm run build 进行打包操作，添加的代码如下：

"scripts": {

"start": "webpack-dev-server --progress --colors --port 3000",

"build": "webpack "

}

整个package.json文件的代码如下：  
{

"name": "itrip",

"version": "1.0.0",

"description": "",

"main": "bundle.js",

"dependencies": {},

"devDependencies": {

"color-convert": "^1.9.0",

"css-loader": "^0.28.4",

"cssesc": "^1.0.0",

"emojis-list": "^2.1.0",

"escape-string-regexp": "^1.0.5",

"fastparse": "^1.1.1",

"has-ansi": "^3.0.0",

"has-flag": "^2.0.0",

"js-base64": "^2.1.9",

"json-stable-stringify": "^1.0.1",

"json5": "^0.5.1",

"loader-utils": "^1.1.0",

"regexpu-core": "^4.1.1",

"schema-utils": "^0.3.0",

"style-loader": "^0.18.2",

"webpack": "^3.4.1",

"webpack-dev-server": "^2.6.1"

},

"scripts": {

"start": "webpack-dev-server --progress --colors --port 3000",

"build": "webpack "

},

"author": "",

"license": "ISC"

}

如果在使用webpack进行打包处理或者本地调试提示报错，可以将以上代码粘贴至package.json文件中，在终端输入 npm install 添加需要的插件，如果还报某某插件未安装输入：npm install \*\*\*\* --save-dev 其中‘\*\*\*\*’为插件名称；