今日目标

1.能够了解模块化的相关规范
2.了解webpack
3.了解使用Vue单文件组件
4.能够搭建Vue脚手架
5.掌握Element-UI的使用

1.模块化的分类

A.浏览器端的模块化

1).AMD(Asynchronous Module Definition,异步模块定义)
代表产品为：Require.js
2).CMD(Common Module Definition,通用模块定义)
代表产品为：Sea.js

B.服务器端的模块化

服务器端的模块化规范是使用CommonJS规范：
1).使用require引入其他模块或者包
2).使用exports或者module.exports导出模块成员
3).一个文件就是一个模块，都拥有独立的作用域

C.ES6模块化

ES6模块化规范中定义：
1).每一个js文件都是独立的模块
2).导入模块成员使用import关键字
3).暴露模块成员使用export关键字

小结：推荐使用ES6模块化，因为AMD，CMD局限使用与浏览器端，而CommonJS在服务器端使用。
ES6模块化是浏览器端和服务器端通用的规范.

2.在NodeJS中安装babel

A.安装babel

打开终端，输入命令：npm install --save-dev @babel/core @babel/cli @babel/preset-env @babel/node
安装完毕之后，再次输入命令安装：npm install --save @babel/polyfill

B.创建babel.config.js

在项目目录中创建babel.config.js文件。
编辑js文件中的代码如下：

```
const presets = [
  ["@babel/env",{
    targets:{
      edge:"17",
      firefox:"60",
      chrome:"67",
      safari:"11.1"
    }
  }]
]
//暴露
module.exports = { presets }
```

C.创建index.js文件

在项目目录中创建index.js文件作为入口文件
在index.js中输入需要执行的js代码，例如：

```
console.log("ok");
```

D.使用npx执行文件

打开终端，输入命令：npx babel-node ./index.js

3.设置默认导入/导出

A.默认导出

```
export default {
    成员A,
    成员B,
    .....
},如下：
let num = 100;
export default{
    num
}
```

B.默认导入

```
import 接收名称 from "模块标识符"，如下：
import test from "./test.js"
```

注意：在一个模块中，只允许使用export default向外默认暴露一次成员，千万不要写多个export default。
如果在一个模块中没有向外暴露成员，其他模块引入该模块时将会得到一个空对象

4.设置按需导入/导出

A.按需导出

```
export let num = 998;
export let myName = "jack";
export function fn = function(){ console.log("fn") }
```

B.按需导入

```
import { num,fn as printFn ,myName } from "./test.js"
//同时导入默认导出的成员以及按需导入的成员
import test,{ num,fn as printFn ,myName } from "./test.js"
```

注意：一个模块中既可以按需导入也可以默认导入，一个模块中既可以按需导出也可以默认导出

5.直接导入并执行代码

```
import "./test2.js";
```

6.webpack的概念

webpack是一个流行的前端项目构建工具，可以解决目前web开发的困境。
webpack提供了模块化支持，代码压缩混淆，解决js兼容问题，性能优化等特性，提高了开发效率和项目的可维护性

7.webpack的基本使用

A.创建项目目录并初始化

创建项目，并打开项目所在目录的终端，输入命令：
npm init -y

B.创建首页及js文件

在项目目录中创建index.html页面，并初始化页面结构：在页面中摆放一个ul，ul里面放置几个li
在项目目录中创建js文件夹，并在文件夹中创建index.js文件

C.安装jQuery

打开项目目录终端，输入命令:
npm install jQuery -S

D.导入jQuery

打开index.js文件，编写代码导入jQuery并实现功能：
import $from$ ” $jquery$ ”;(function(){
($” li:odd ”).$css$(” $background$ ”,” $cyan$ ”);("li:odd").css("background","pink");
})
注意：此时项目运行会有错误，因为import $ from "jquery";这句代码属于ES6的新语法代码，在浏览器中可能会存在兼容性问题
所以我们需要webpack来帮助我们解决这个问题。

E.安装webpack

1).打开项目目录终端，输入命令:
npm install webpack webpack-cli -D
2).然后在项目根目录中，创建一个 webpack.config.js 的配置文件用来配置webpack
在 webpack.config.js 文件中编写代码进行webpack配置，如下：
module.exports = {
mode:"development"//可以设置为development(开发模式)，production(发布模式)
}
补充：mode设置的是项目的编译模式。
如果设置为development则表示项目处于开发阶段，不会进行压缩和混淆，打包速度会快一些
如果设置为production则表示项目处于上线发布阶段，会进行压缩和混淆，打包速度会慢一些
3).修改项目中的package.json文件添加运行脚本dev，如下：
"scripts":{
"dev":"webpack"
}
注意：scripts节点下的脚本，可以通过 npm run 运行，如：
运行终端命令：npm run dev
将会启动webpack进行项目打包
4).运行dev命令进行项目打包，并在页面中引入项目打包生成的js文件
打开项目目录终端，输入命令:
npm run dev

等待webpack打包完毕之后，找到默认的dist路径中生成的main.js文件，将其引入到html页面中。
浏览页面查看效果。

8.设置webpack的打包入口/出口

在webpack 4.x中，默认会将src/index.js 作为默认的打包入口js文件
默认会将dist/main.js 作为默认的打包输出js文件
如果不想使用默认的入口/出口js文件，我们可以通过改变 webpack.config.js 来设置入口/出口的js文件，如下：

```
const path = require("path");
module.exports = {
  mode:"development",
  //设置入口文件路径
  entry: path.join(dirname,"./src/xx.js"),
  //设置出口文件
  output:{
    //设置路径
    path:path.join(dirname,"./dist"),
    //设置文件名
    filename:"res.js"
  }
}
```

9.设置webpack的自动打包

默认情况下，我们更改入口js文件的代码，需要重新运行命令打包webpack，才能生成出口的js文件
那么每次都要重新执行命令打包，这是一个非常繁琐的事情，那么，自动打包可以解决这样繁琐的操作。
实现自动打包功能的步骤如下：

A.安装自动打包功能的包:webpack-dev-server

```
npm install webpack-dev-server -D
```

B.修改package.json中的dev指令如下：

```
"scripts":{
  "dev":"webpack-dev-server"
}
```

C.将引入的js文件路径更改为：
D.运行npm run dev，进行打包
E.打开网址查看效果：http://localhost:8080

> 注意：webpack-dev-server自动打包的输出文件，默认放到了服务器的根目录中.

补充：
在自动打包完毕之后，默认打开服务器网页，实现方式就是打开package.json文件，修改dev命令：

```
"dev": "webpack-dev-server --open --host 127.0.0.1 --port 9999"
```

10.配置html-webpack-plugin

使用html-webpack-plugin 可以生成一个预览页面。
因为当我们访问默认的 http://localhost:8080/的时候，看到的是一些文件和文件夹，想要查看我们的页面
还需要点击文件夹点击文件才能查看，那么我们希望默认就能看到一个页面，而不是看到文件夹或者目录。

实现默认预览页面功能的步骤如下：

A.安装默认预览功能的包:html-webpack-plugin

npm install html-webpack-plugin -D

B.修改webpack.config.js文件，如下：

```
//导入包
const HtmlWebpackPlugin = require("html-webpack-plugin");
//创建对象
const htmlPlugin = new HtmlWebpackPlugin({
  //设置生成预览页面的模板文件
  template:"./src/index.html",
  //设置生成的预览页面名称
  filename:"index.html"
})
```

C.继续修改webpack.config.js文件，添加plugins信息：

```
module.exports = {
  ......
  plugins:[ htmlPlugin ]
}
```

11.webpack中的加载器

通过loader打包非js模块：默认情况下，webpack只能打包js文件，如果想要打包非js文件，需要调用loader加载器才能打包

loader加载器包含：

1).less-loader
2).sass-loader
3).url-loader:打包处理css中与url路径有关的文件
4).babel-loader:处理高级js语法的加载器
5).postcss-loader
6).css-loader,style-loader

注意：指定多个loader时的顺序是固定的，而调用loader的顺序是从后向前进行调用

A.安装style-loader,css-loader来处理样式文件

1).安装包

```
npm install style-loader css-loader -D
```

2).配置规则：更改webpack.config.js的module中的rules数组

```
module.exports = {
    ......
    plugins:[ htmlPlugin ],
    module : {
        rules:[
            {
                //test设置需要匹配的文件类型，支持正则
                test:/\.css$/,
                //use表示该文件类型需要调用的loader
                use:['style-loader','css-loader']
            }
        ]
    }
}
```

B.安装less,less-loader处理less文件

1).安装包

```
        npm install less-loader less -D
    2).配置规则：更改webpack.config.js的module中的rules数组
    module.exports = {
        ......
        plugins:[ htmlPlugin ],
        module : {
            rules:[
                {
                    //test设置需要匹配的文件类型，支持正则
                    test:/\.css$/,
                    //use表示该文件类型需要调用的loader
                    use:['style-loader','css-loader']
                },
                {
                    test:/\.less$/,
                    use:['style-loader','css-loader','less-loader']
                }
            ]
        }
    }
C.安装sass-loader,node-sass处理less文件
    1).安装包
        npm install sass-loader node-sass -D
    2).配置规则：更改webpack.config.js的module中的rules数组
    module.exports = {
        ......
        plugins:[ htmlPlugin ],
        module : {
            rules:[
                {
                    //test设置需要匹配的文件类型，支持正则
                    test:/\.css$/,
                    //use表示该文件类型需要调用的loader
                    use:['style-loader','css-loader']
                },
                {
                    test:/\.less$/,
                    use:['style-loader','css-loader','less-loader']
                },
                {
                    test:/\.scss$/,
                    use:['style-loader','css-loader','sass-loader']
                }
            ]
        }
    }

    补充：安装sass-loader失败时，大部分情况是因为网络原因，详情参考：
    https://segmentfault.com/a/1190000010984731?utm_source=tag-newest

D.安装post-css自动添加css的兼容性前缀（-ie-,-webkit-）
1).安装包
    npm install postcss-loader autoprefixer -D
2).在项目根目录创建并配置postcss.config.js文件
const autoprefixer = require("autoprefixer");
module.exports = {
    plugins:[ autoprefixer ]
}
```

3).配置规则：更改webpack.config.js的module中的rules数组

```
module.exports = {
    ......
    plugins:[ htmlPlugin ],
    module : {
        rules:[
            {
                //test设置需要匹配的文件类型，支持正则
                test:/\.css$/,
                //use表示该文件类型需要调用的loader
                use:['style-loader','css-loader','postcss-loader']
            },
            {
                test:/\.less$/,
                use:['style-loader','css-loader','less-loader']
            },
            {
                test:/\.scss$/,
                use:['style-loader','css-loader','sass-loader']
            }
        ]
    }
}
```

E.打包样式表中的图片以及字体文件

在样式表css中有时候会设置背景图片和设置字体文件，一样需要loader进行处理

使用url-loader和file-loader来处理打包图片文件以及字体文件

1).安装包

```
    npm install url-loader file-loader -D
```

2).配置规则：更改webpack.config.js的module中的rules数组

```
module.exports = {
    ......
    plugins:[ htmlPlugin ],
    module : {
        rules:[
            {
                //test设置需要匹配的文件类型，支持正则
                test:/\.css$/,
                //use表示该文件类型需要调用的loader
                use:['style-loader','css-loader']
            },
            {
                test:/\.less$/,
                use:['style-loader','css-loader','less-loader']
            },
            {
                test:/\.scss$/,
                use:['style-loader','css-loader','sass-loader']
            },{
                test:/\.jpg|png|gif|bmp|ttf|eot|svg|woff|woff2$/,
                //limit用来设置字节数，只有小于limit值的图片，才会转换
                //为base64图片
                use:"url-loader?limit=16940"
            }
        ]
    }
}
```

F.打包js文件中的高级语法：在编写js的时候，有时候我们会使用高版本的js语法
有可能这些高版本的语法不被兼容，我们需要将之打包为兼容性的js代码
我们需要安装babel系列的包
A.安装babel转换器

```
    npm install babel-loader @babel/core @babel/runtime -D
```

B.安装babel语法插件包

```
    npm install @babel/preset-env @babel/plugin-transform-runtime @babel/plugin-proposal-class-properties -D
```

C.在项目根目录创建并配置babel.config.js文件

```
    module.exports = {
        presets:["@babel/preset-env"],
        plugins:[ "@babel/plugin-transform-runtime", "@babel/plugin-proposal-class-properties" ]
    }
```

D.配置规则：更改webpack.config.js的module中的rules数组

```
module.exports = {
    ......
    plugins:[ htmlPlugin ],
    module : {
        rules:[
            {
                //test设置需要匹配的文件类型，支持正则
                test:/\.css$/,
                //use表示该文件类型需要调用的loader
                use:['style-loader','css-loader']
            },
            {
                test:/\.less$/,
                use:['style-loader','css-loader','less-loader']
            },
            {
                test:/\.scss$/,
                use:['style-loader','css-loader','sass-loader']
            },{
                test:/\.jpg|png|gif|bmp|ttf|eot|svg|woff|woff2$/,
                //limit用来设置字节数，只有小于limit值的图片，才会转换
                //为base64图片
                use:"url-loader?limit=16940"
            },{
                test:/\.js$/,
                use:"babel-loader",
                //exclude为排除项，意思是不要处理node_modules中的js文件
                exclude:/node_modules/
            }
        ]
    }
}
```

12.Vue单文件组件

传统Vue组件的缺陷：
全局定义的组件不能重名，字符串模板缺乏语法高亮，不支持css(当html和js组件化时，css没有参与其中)
没有构建步骤限制，只能使用H5和ES5，不能使用预处理器（babel）
解决方案：
使用Vue单文件组件，每个单文件组件的后缀名都是.vue

每一个Vue单文件组件都由三部分组成
1).template组件组成的模板区域
2).script组成的业务逻辑区域
3).style样式区域

代码如下：

```
<template>

    组件代码区域

</template>

<script>

    js代码区域

</script>

<style scoped>

    样式代码区域

</style>
```

补充：安装Vetur插件可以使得.vue文件中的代码高亮

配置.vue文件的加载器
A.安装vue组件的加载器
  npm install vue-loader vue-template-compiler -D
B.配置规则：更改webpack.config.js的module中的rules数组

```
    const VueLoaderPlugin = require("vue-loader/lib/plugin");
    const vuePlugin = new VueLoaderPlugin();
    module.exports = {
        ......
        plugins:[ htmlPlugin, vuePlugin  ],
        module : {
            rules:[
                ...//其他规则
                    {
                        test:/\.vue$/,
                        loader:"vue-loader",
                    }
        ]
    }
}
```

13.在webpack中使用vue

上一节我们安装处理了vue单文件组件的加载器，想要让vue单文件组件能够使用，我们必须要安装vue并使用vue来引用vue单文件组件。

A.安装Vue

```
npm install vue -S
```

B.在index.js中引入vue：import Vue from "vue"

C.创建Vue实例对象并指定el，最后使用render函数渲染单文件组件

```
const vm = new Vue({
    el:"#first",
    render:h=>h(app)
})
```

14.使用webpack打包发布项目

在项目上线之前，我们需要将整个项目打包并发布。

A.配置package.json

```
"scripts":{
    "dev":"webpack-dev-server",
    "build":"webpack -p"
}
```

B.在项目打包之前，可以将dist目录删除，生成全新的dist目录

15.Vue脚手架

Vue脚手架可以快速生成Vue项目基础的架构。

A.安装3.x版本的Vue脚手架：

```
npm install -g @vue/cli
```

B.基于3.x版本的脚手架创建Vue项目：

1).使用命令创建Vue项目

命令：vue create my-project

选择Manually select features(选择特性以创建项目)

勾选特性可以用空格进行勾选。

是否选用历史模式的路由：n

ESLint选择：ESLint + Standard config

何时进行ESLint语法校验：Lint on save

babel，postcss等配置文件如何放置：In dedicated config files(单独使用文件进行配置)

是否保存为模板：n

使用哪个工具安装包：npm

2).基于ui界面创建Vue项目

命令：vue ui

在自动打开的创建项目网页中配置项目信息。

3).基于2.x的旧模板，创建Vue项目

```
npm install -g @vue/cli-init
vue init webpack my-project
```

C.分析Vue脚手架生成的项目结构

node_modules:依赖包目录

public：静态资源目录

src：源码目录

src/assets:资源目录

src/components：组件目录

src/views:视图组件目录

src/App.vue:根组件
src/main.js:入口js
src/router.js:路由js
babel.config.js:babel配置文件
.eslintrc.js:

16.Vue脚手架的自定义配置

A.通过 package.json 进行配置 [不推荐使用]

```
"vue":{
  "devServer":{
    "port":"9990",
    "open":true
  }
}
```

B.通过单独的配置文件进行配置，创建vue.config.js

```
module.exports = {
  devServer:{
    port:8888,
    open:true
  }
}
```

17.Element-UI的基本使用

Element-UI:一套基于2.0的桌面端组件库
官网地址：http://element-cn.eleme.io/#/zh-CN
A.安装：

```
npm install element-ui -S
```

B.导入使用：

```
import ElementUI from "element-ui";
import "element-ui/lib/theme-chalk/index.css";
Vue.use(ElementUI)
```