NODE.JS

node.js是基于V8引擎运行JS的环境,它不是一门语言.他是一个工具,甚至你可以理解为他是一个谷歌浏览器

1.为什么我们成为他后台语言?

> 客户端通过发送请求到服务器上,服务器也需要编写相关程序,把客户端的请求的内容准备好,然后返回给客户端,
> 我们以前的时候使用的是java或者php等语言来编写相关的后台程序,但是现在我们也可以使用JS来编写后台的程序(JS不仅可以写前端程序也可以实现后台程序,所以说我们的JS是全栈语言)
> 我们写好的js作为后台程序需要运行,那就需要能识别JS代码的环境,此时node就诞生了,我们只需要在服务器上装一个node工具,让js在这个node环境中编译执行即可

2.安装node

https://nodejs.org/en/ 英文
http://nodejs.cn/ 中文
查看安装完成 node -v

3.node.js烦人基础知识

node.js优势

- 基于V8引擎渲染和解析代码(快)
- 基于事件驱动的非阻塞I/O操作
- 采用异步单线程开发
- npm包管理器是全球最大的开源库
  https://www.npmjs.com/
- 对于前端开发上手快

node中的模块

```
node中的模块分为三种
- 内置模块:node环境(工具)中自带的,类似于浏览器中也会自带一些自己的方法
- 自定义模块:开发人员自己的写的
- 第三方模块:别人写好的我们拿过来使用(类似于我们引入一个JQ)
我们需要的第三方模块都可以在npm包管理器中找到,下载下来使用即可
以后再装模块的时候都是使用npm install XX
```

npm包管理器以及第三方模块

npm install xxx -g 将模块xxx安装到全局环境中
npm install xxx -global
注意:在浏览器中最大的全局对象是window,但是node中没有window,全局对象是global

npm install xxx 安装到当前项目下
npm install xxx --save-dev 不仅将模块安装到当前项目中,而且将安装的信息记录到package.json清单中,生成一条开发环境依赖项
npm install xxx --save 跟上一个差不多,只不过生成一条生产环境依赖项
npm uninstall xxx -g/--save-dev/--save 删除指定目下的模块

npm install jquery@1.11.3 安装模块的时候指定版本号

1.安装到全局和项目下的区别

安装到全局

例如安装less

```
npm install less -g
```

安装到全局环境中可以使用dos命令来操作,但是你只可以使用命令操作,不可以导入到自己的JS代码中

安装到当前项目中

```
npm install less
```

安装完成后会在当前项目中增加了一个文件夹node_modules 此文件夹中就会多一个模块less,以后再装模块的时候就会自动加在这文件夹node_modules中

注意:安装到当前项目中是不可以使用dos命令操作的(默认情况下),但是我们在JS代码可以导入进来,使用require 导入其他模块到JS文件中,在这就是文件中就可以使用导入进来的模块的方法了

2.能否有办法既能使用命令操作还可以导入到JS中?

真实项目中.开发的时候我们很少讲模块安装在全局中,因为装在全局很肯能会导致版本的冲突,一般都是装在本项目中

在本项目中如何使用命令来配置模块

npm init -y
在本地项目中生成一个package.json文件,这个文件中是此项目的配置信息
-y写上就是在这个package.json文件中自动生成配置信息,不需要你手动写入,比较方便快捷

需要在package的scipts这个对象中加入自定义的命令zxt

```
  "scripts": {
//    "test": "echo \"Error: no test specified\" && exit 1",
    //zxt:自定义的属性名
    //属性值:当你在dos命令窗口执行 zxt的时候实际上执行的就是属性值中的命令
    "zxt":"lessc less/index.less less/index.min.css -x"
  },
```

接下来在 当前目录下执行命令 npm run zxt

开发环境和生产环境

> 开发环境:项在你本地开发的时候,你需要的模块就叫做开发依赖项
> 生产环境:项目完成,部署到服务器上,这个时候需要的模块就是生产依赖项
> 例如less:为了方便开发人员编写样式创造的一个工具,这个模块只是在开发的时候有些开发人员使用的,最终部署到服务器上的时候任然是编译好的css文件,所以less 模块只是开发依赖,线上是不需要安装的

npm install less --save-dev 安装less模块并把模块信息保存到配置信息package.json中放在开发依赖项

npm install bootstrap --save 安装bootstrap模块并把模块信息保存到配置信息package.json中放在生成依赖项

```
  "dependencies": { 生产依赖项
    "bootstrap": "^3.3.7"
  },
  "devDependencies": {开发依赖项
    "less": "^2.7.3"
  }
}
```

3.为啥要设置依赖项

真实项目中肯定是多人协作开发项目,比如使用git管理项目代码以及团队的协作处理
比如每个人都在本地开发的时候导入了很多模块有开发依赖也有生产依赖,每个一个都在node_modules中,这包很大,当项目完成了,此时我们需要将每一个开发人员的项目合并到一起,如果之前有记录,记录了每一个开发人员的配置信息,模块的使用信息,我们只需要将这个记录传上去,此时package就起了大作用了,当别人拿到这个项目的时候只需要npm install一下就会把当前依赖的模块自动装上,我们管这个操作叫做`跑环境`

4.安装指定版本的模块

查询一下模块的版本信息
`npm view webpack` 查看webpack的版本信息
`npm view webpack >version.webpack` 由于内容太多了我们看不方便,我们将信息输出到version.webpack文件中(文件名字可以自定义)
`npm install webpack@2.7.0 --save-dev`

node中的自定义模块

使用js写后台程序,基于node环境运行,我们创建一个js文件这个文件中我们处理了一个逻辑问题(对比我们之前封装的方法),在另一个js文件如果我们也想使用这个个逻辑,就不需要再一遍,直接将之前的js文件导入进来,
所以在node环境中可以实现模块之间的调用

1.require 当前模块中国导入其他模块

```
require("XXX");
```

这样的写法导入的是第三方模块或者是内置模块,就是你之前安装在node_modules中的模块

例如 `require("less");`

当你使用这个方式导入的时候,查找的顺序是先看第三方模块也就是node_modules中有没有这个模块,没有的话然后再去看内置模块,如果还没有就报错了

```
require("./xxx");
require("../x/xxx");
```

导入模块的时候加上路径的话导入的是自定义模块,但是路径得对,不对的话肯定报错了

一般我们会使用一个变量接受一下

```
const temp=require("./temp.js");
```

可以将.js省略

```
const temp=require("./temp");
```

module.exports

module:是node环境中天生自带的一个对象,进行node模块管理的

对象下面以有个属性exports,意思是把这个模块的某些方法导出,提供给其他模块使用

练习

有三个自定义模块 ZF1 ,ZF2,ZF3
ZF1中提供一个sum;实现任意数求和
ZF2中提供一个avg方法;实现求平均数,去掉一个最大值和最小值然后求平均数,这里必须要用到ZF1中的sum方法
ZF3中,准备一些数字,实现ZF2中的avg方法求平均数