## 自定义模块并且上传npm官网

1. 模块里面：

require——引入

export——输出

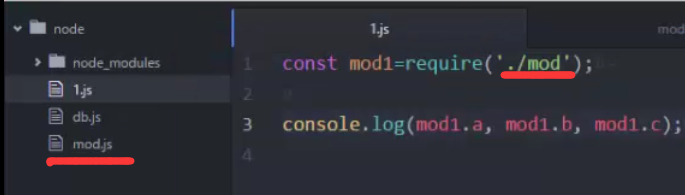
module-exports——批量输出

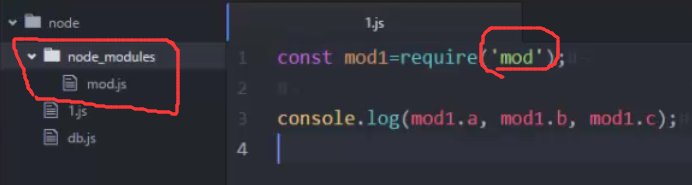
1. npm：
   1. 帮助下载模块
   2. 自动解决依赖
2. 一个JS文件要引用其他模块需满足下面的语法规则（common.js）：

Const model1 = require（“模块名”）；

1. 模块名这里有讲究，

如果模块名有“./”，从当前目录找

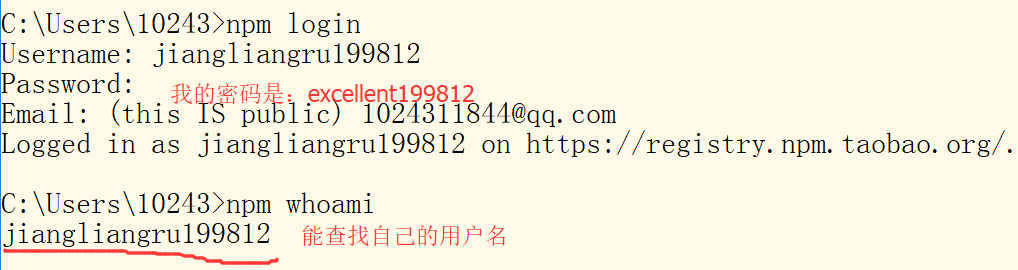
如果没有“./”，从系统模块或者是node\_modules文件里去找



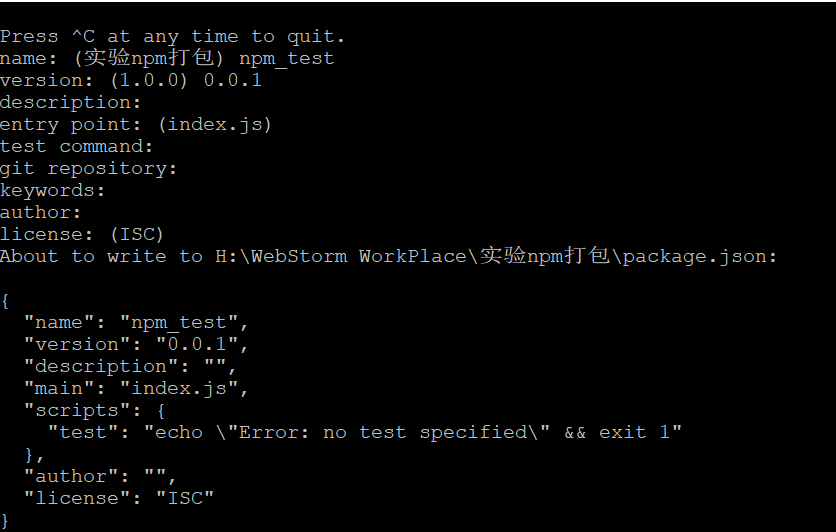
5. 若有同名模块，则系统模块优先：如果自定义一个‘http’模块，因为和系统模块‘http’相冲突，所以系统模块‘http’优先

### 正片：

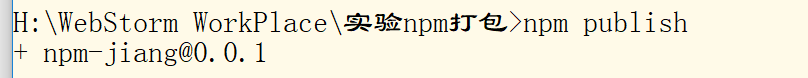
1. 登陆自己的npm账号：



1. npm init：初始化包信息

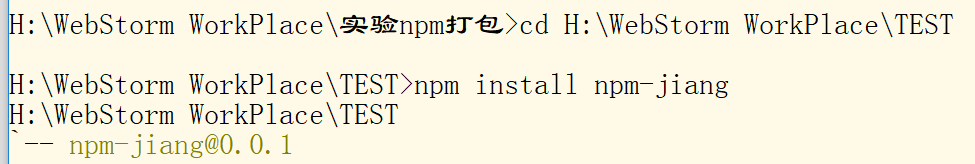


1. 发布npm包



1. 最后可以在其他主文件里通过npm install 模块名 来下载自定义模块，然后用

require语法来引入使用自定义模块



1. 更新npm包：npm update 模块名
2. 删除npm包：需要先进入到定义npm包的文件目录下，

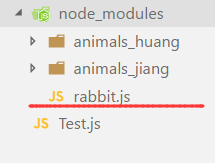
输入npm unpublish –force

\*\*\*\*\*\*\*对node\_modules的理解（非常重要）

1.首先先明确一点，node\_nodules文件夹里的文件类型只能有两种，一个是文件夹，一个是文件（JS文件），因为JS模块的定义就是一个JS文件对应一个JS模块,而且通过require导入的也是模块（也就是一个JS文件），所以在别的JS文件里导入模块是很容易的，这里的

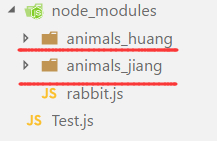
let rabbit = **require**('rabbit');

导入方式的前提是被导入的JS文件必须存在于node\_modules文件夹中，不然导入必须按照JS文件原来的路径导入（文章开头有说）





2.那么难点来了，对于node\_modules文件夹里的文件夹该如何理解呢



首先必须明确，这两个是文件夹，也就是说不是模块，不是模块，不是模块（重要的事说三遍），所以说在哦不进行过任何处理的情况下使用下面语句是完全错误的

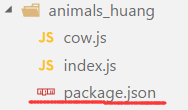
let animals\_jiang = **require**('animals\_jiang');



不管你这个文件夹里面有什么JS文件，首先你是导入一个文件夹，这本来就不符合模块导入规则（导入的是JS文件，不是文件夹），

那么我就是想这样子导入，该如何不报错呢，

首先必须先通过npm init命名来创建一个package.json文件



这个package.json文件非常重要，如果要想导入文件夹，这个package.json文件是必要的

{

"name": "animals\_huang",

"version": "1.0.0",

"description": "",

***"main": "index.js"***,

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

在这个package.json文件里最重要的一个东西就是“main”这一行，他后面跟了一个值（json文件本身是一个键值对的集合），这个index.js是一个入口文件，那么如何理解呢？

let animals\_jiang = **require**('animals\_jiang');

有了这个***"main": "index.js"，***上面的导入语句就等价于下面的导入语句

let animals\_jiang = **require**('index');

那么如果把index.js改为cow.js，那么结果如何呢？

那么导入语句就等同于

let animals\_huang = **require**('cow');

那么接下来就分别对cow.js和index.js的文件内容来具体分析

***"main": "index.js"***

***和***

***"main": "cow.js"的区别***

先来看cow.js

也就是说

let animals\_jiang = **require**('animals\_jiang');

等价于

let animals\_huang = **require**('cow');

module.exports = {

**talk**: function () {

**console**.**log**("mou~");

}

}

*// exports.talk = function(){*

*// console.log("mou~")*

*// }*

先别去管注释的部分，

先看这行

module.exports = {

**talk**: function () {

**console**.**log**("mou~");

}

}

这代表了cow.js文件暴露出了一个对象（注意是对象而不是方法，只不过对象里面有一个属性，他的数据类型为Function，这点很重要）

通过

**console**.**log**(animals\_huang);

来看看animals\_huang是什么样子的类型

结果输出一个对象类型：

{ talk: [Function: talk] }

这跟我们刚才说的module.exports暴露出去的就是一个对象的结论完美契合。也就是说animals\_huang是个对象

所以

animals\_huang.**talk**();

输出的结果是：mou~

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

那么下面我们把刚才被注释掉的代码的注释打开

exports.**talk** = function(){

**console**.**log**("mou~")

}

再通过

**console**.**log**(animals\_huang);

看看animals\_huang是什么样子的类型

{ talk: [Function] }

貌似没区别……animals\_huang同样都是一个对象

animals\_huang.**talk**()

输出结果还是：mou~

再来看index.js

也就是说

let animals\_huang = **require**('animals\_huang');

等价于

let animals\_huang = **require**('index');

exports.cow = **require**('./cow');

但先让我们看看如果index.js里没有这行代码会出什么问题

**console**.**log**(animals\_huang)

输出：{} 是个空对象

所以说执行

animals\_huang.**talk**()

必然报错

TypeError: animals\_huang.talk is not a function

空对象哪来的方法

所以说

exports.cow = **require**('./cow');

这行代码的作用就是cow.js里的代码导入到index.js文件里

导入了这行代码以后执行

**console**.**log**(animals\_huang)

执行结果为：

{ cow: { talk: [Function] } }

怎么理解这个结果呢：

因为在

**require**('./cow')

的执行下导入了一个对象（因为cow.js通过exports导出了一个对象），这个对象的值为

{ talk: [Function] }

而exports.cow

导出了一个对象，这个对象的值为

{ cow: { talk: [Function] } }

当然如果你乐意，你也可以把

exports.cow = **require**('./cow');

改为

module.exports = {

cow: **require**('./cow')

}

效果都一样

所以说如果想输出mou~

那么执行语句就是

animals\_huang.cow.**talk**()

最后一个问题：

如果文件结果是下面这样的



文件夹里面由套有一个文件夹，然后JS文件都在内层文件夹里面，那么如何分析呢

道理都一样的呀

只需在package.json里面的

***“main”键值对中设置对应的文件***

以上面文件夹格式为例

"main": "index.js"

Index.js里文件：

exports.cat = **require**('./animals/cat');

exports.dog = **require**('./animals/dog');

不都是一个道理吗~~~

所以总结，在node包里面

1.npm init创建一个package.json文件（最重要）

2.然后设置“main”的值就可以了（最最重要）