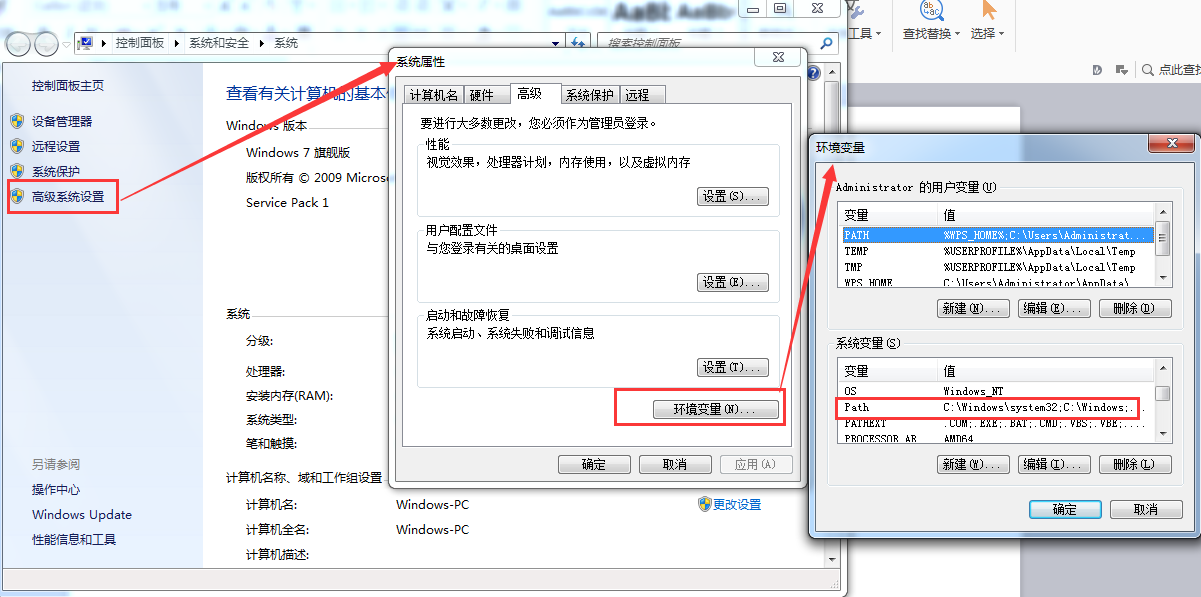
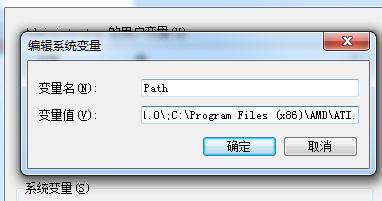
**Node的环境变量如何配置？**

计算机 => 属性 => 高级系统设置 =>系统属性中点环境变量 => 系统变量中找到path

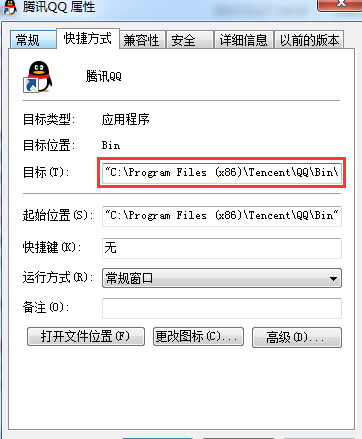


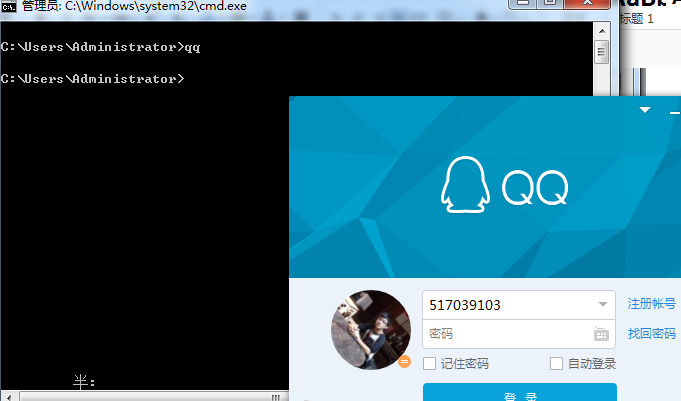
双击path =>

如果要加入变量值，需要在前一个后面加上;代表上一个结尾。

如我们要把QQ加入：复制路径（不要前后的””号、不要xxx.exe）得到：

C:\Program Files (x86)\Tencent\QQ\Bin\

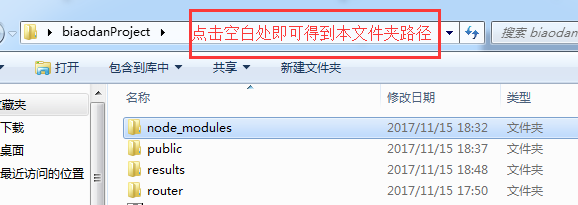


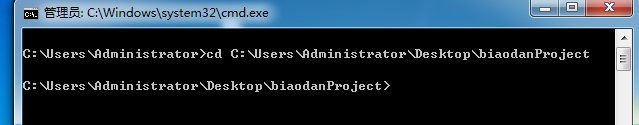
此时cmd中输入qq：，即可启动QQ程序。

所以再cmd中直接输入node命令，其实也是启动了node.exe

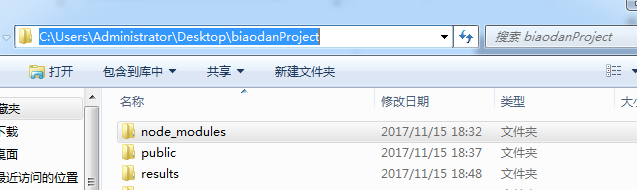
**启动cmd，对应文件路径的三种方式：**

1. 得到文件路径，在cmd中 => cd 粘贴文件路径





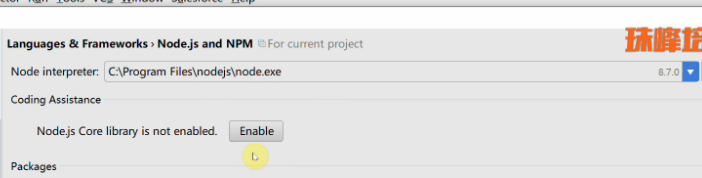
1. 点击后直接在文件夹路径上输入cmd



1. 在文件夹里 shift+鼠标右键选择此处打开命令窗口。

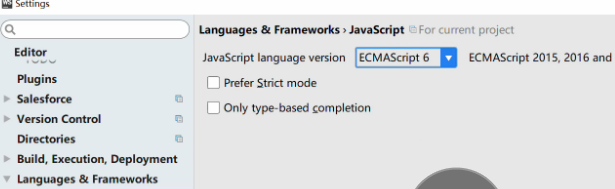
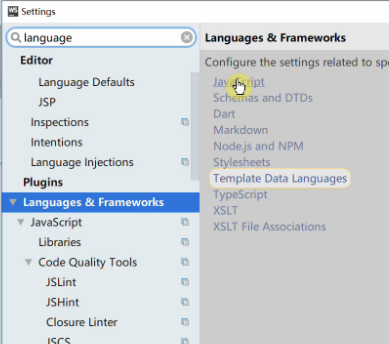
**在webstrom中设置node运行环境：**

设置(setting) => 搜索npm => 选择路径找到node.exe的正确路径位置=>点击Enable会配置node提示包，这样在webstrom中的js文件，右键菜单会出现一个run xx.js，点击run就会启动控制台(相当于cmd中运行Node)。(run下面有个debug,相当于JS中的打断点，走到那会停)



**在webstrom中配置ES6的运行环境：**

Setting => 输入language =>选中javascript =>选中ES6



**知识点：**

-主线程是单线程（异步）callback,将后续的逻辑写成函数，传入到当前执行的函数中，当执行的函数得到了结果后，执行传入的函数（回调函数）

-一个服务员同时为五个桌子的顾客服务（异步）

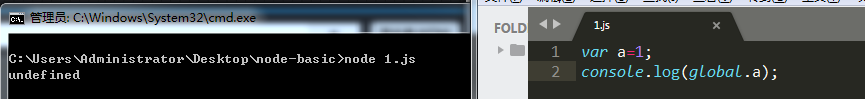
-阻塞不能异步（阻塞是针对内核说的）

-i/o操作，读写操作，异步读写（能用异步绝不用同步）

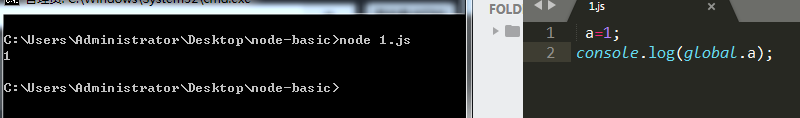
-event-driven 事件驱动 (相当于发布订阅模式) 执行时是事件环操作，执行当前，记录下一次要做的，执行完当前，执行下次一次，并记录下下一次，一环套一环。

在js文件中打印console.log(this) =>这个this不是global,是一个空对象=>{},但是在cmd中this就会输出global。Node自带模块化功能，一个js文件就是一个模块，模块this不是global.

每个文件都有局部作用域（就是一个闭包，外面套了一层自执行函数），不会将属性挂载到global上。



输出”undefined”。就是因为1.js是一个独立的作用域，并且里面有了var 关键字声明了a,global.a,因为这个独立作用域就是个闭包，global自然就访问不到。



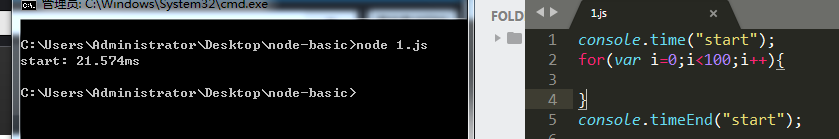
当a不是在私有作用域里声明过后，a就是“自由”的，会往上级找，因为没var 不会受到闭包的影响，就会冲破1.js的块级作用域，找到global,相当于是挂载到global上的一个a属性，所以能输出1.

全局变量，在所有模块中均可使用，可以不声明直接使用。就是直接挂在在global上的属性。如：

Buffer(缓存区)=>文件读写到Buffer中

clearImmediate 清除立即 setImmediate 设置立即

console如下面打印时差：算一个循环跑了多久，要保证()中的标识是一致的才能算出来

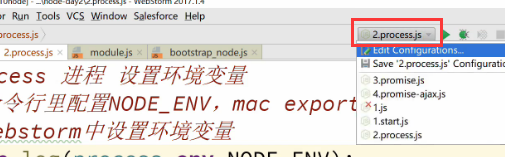


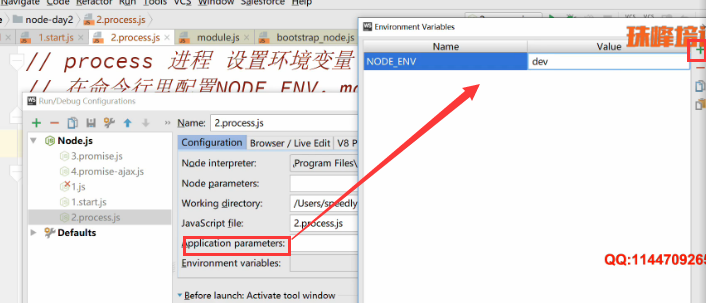
process(进程)=>设置环境变量，来区分当前是开发还是上线。

1.在node中设置



1. 在webstrom中设置





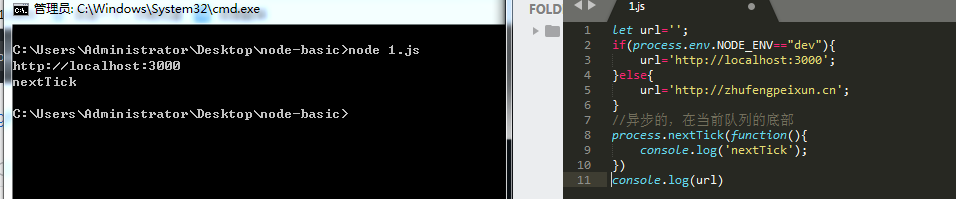
如果代码放在服务器上，那就没有此环境变量，取不到可以走上线环境：



如果要放上线了，只需要把NODE\_ENV原来的值dev，随便改个值，就走另一个url地址了

一下说的都是异步的，在当前队列的底部

process.nextTick();

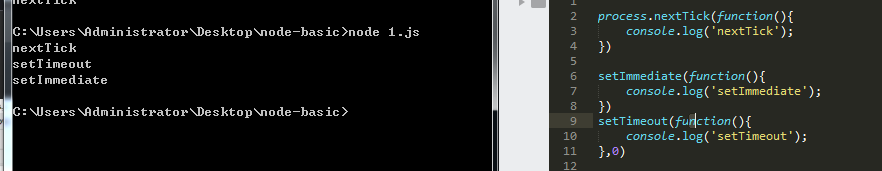


Process.nextTick() setImmediate() setTimeout() 这三个都是异步：

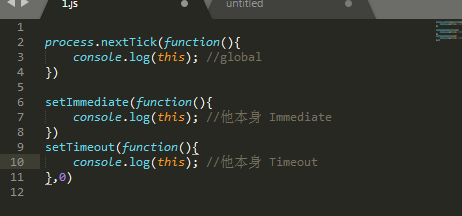
相对着急的用process.nextTick

不设置时间的用 setImmediate

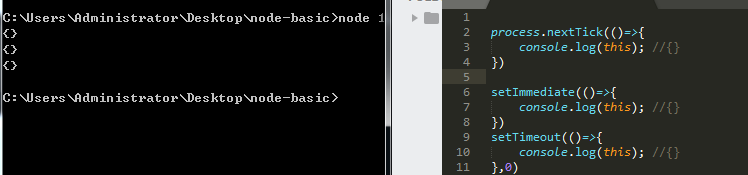
确定时间的用 setTimeout

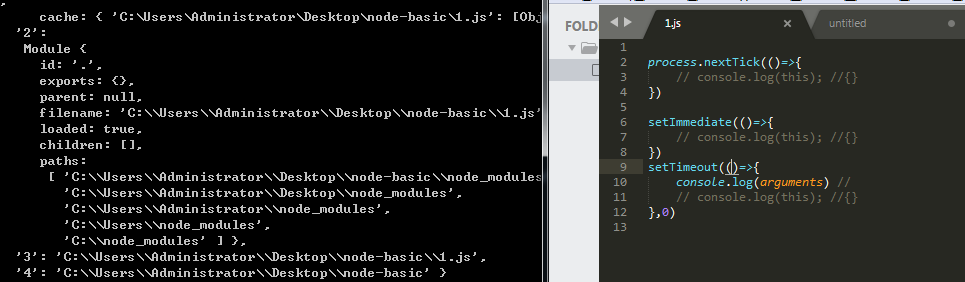


This问题：



换成箭头函数：箭头函数中没有this指向，也没有arguments



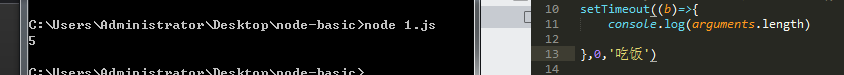
如果把arguments打印出来：

发现有五个(0,1,2,3,4)参数，但是这五个参数并不是setTimeout的，因为这个JS在node下运行本身就是一个闭包，是其父级(自执行函数/闭包)的。

那么给箭头函数如何传参呢？

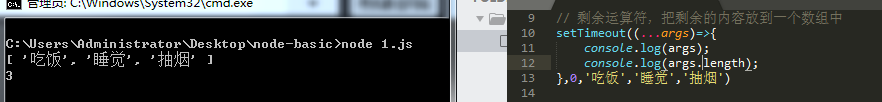


并且此时再输出arguments.length,但这个五个还是其父级的

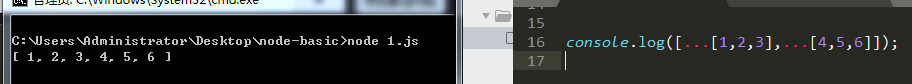


我们要用到ES6的语法：剩余运算符：把剩余的内容放到一个数组中，此时就得到了箭头函数中传递的参数和个数。

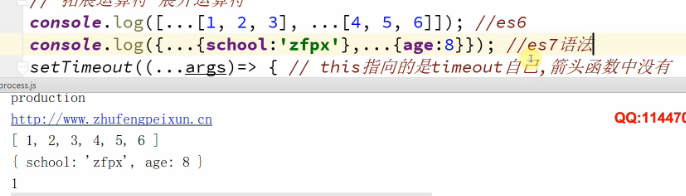
运算符写在形参中=>叫做剩余运算符



扩展运算符 展开运算符，用法：把两个数组合并，放入一个数组，在一个大数组中有两个小数组，给两个数组前面加上...，就是要把这个数组展开放入大数组中



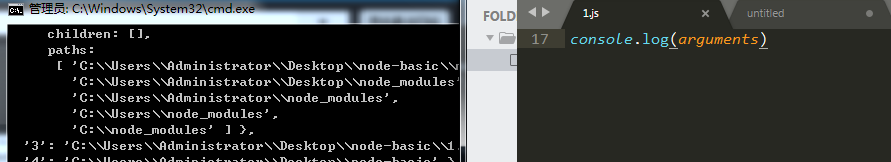
还可以给对象用，是需要node高版本支持，ES7语法：



------------------------------------------------------------------------------------------------------------------------------

刚才说了上面的arguments打印出来了，并且有五个参数

那么是如何验证是父级的arguments? :直接在1.js中输出arguments,我们发现1.js中就只有打印arguments这一句话，一般arguments是要在函数体中，但确实是输出出来了，那就说明1.js是有一个自执行函数(闭包)的块级作用域的。



并且这五个参数分别是：

Exports / require / module / \_\_filename / \_\_dirname

模块化(闭包)：1.低耦合，高内聚 2.方便维护，防止代码冲突（命名冲突）

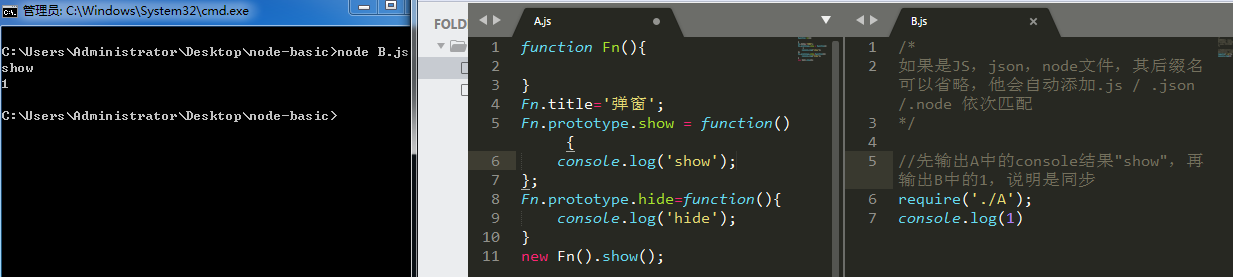
单例模式：不能保证一定不冲突(重名),写成很特殊的名字很长的话，导致调用过程长

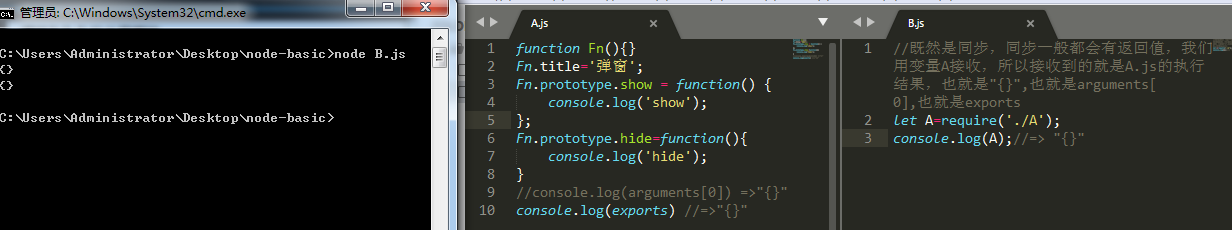
如: var hl18103747258={ a:’xxx’}; 调用时：hl18103747258.a

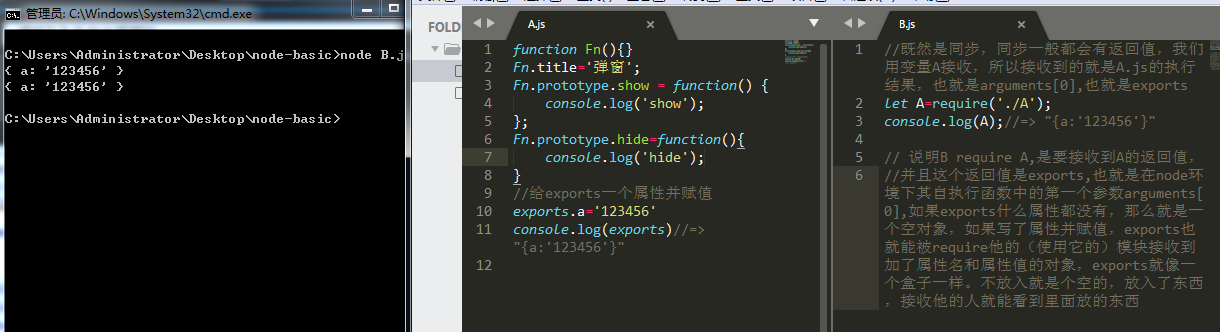
Node基于common.js 文件的读写，node天生自带模块化（天生套了一个闭包）

1. 定义如何创建一个模块 =>一个js文件就是一个模块
2. 如何使用一个模块 =>使用一个文件只需要require一个文件
3. 如何导出一个模块 => 用exports 或 module.exports 导出

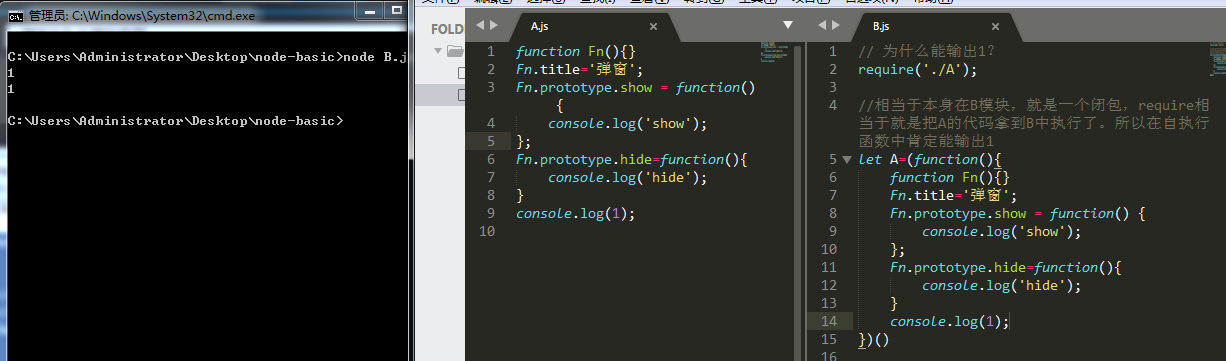
如有两个JS文件A和B，B调用A的话：

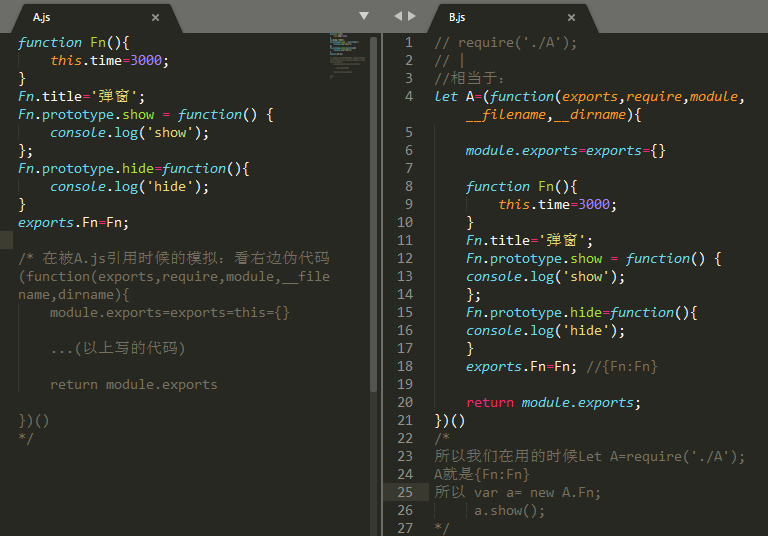


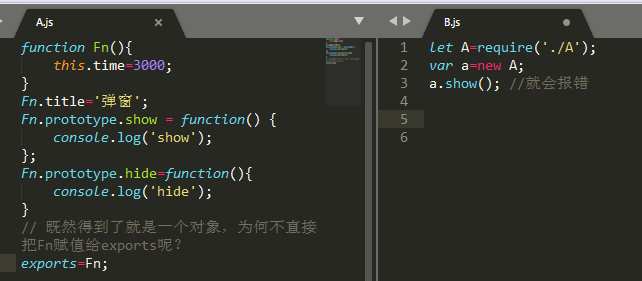




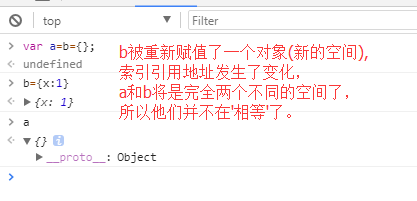
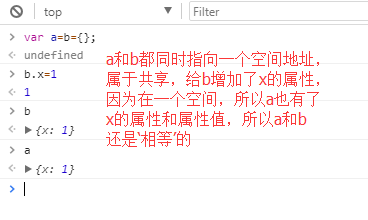








看以下两个小例子，空间的引用问题

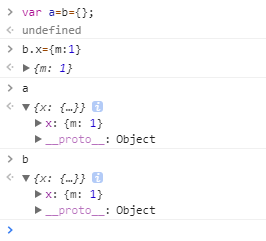


上面之所以不能直接把Fn给exports就是因为Fn本身就是一个空间，直接给了exports就相当于指向Fn了，而不能与modules.exports 、this 共享一个空间地址了。而module.exports就还是{}，所以new 一个空对象肯定会报错啊



综上：引用有两种方式：

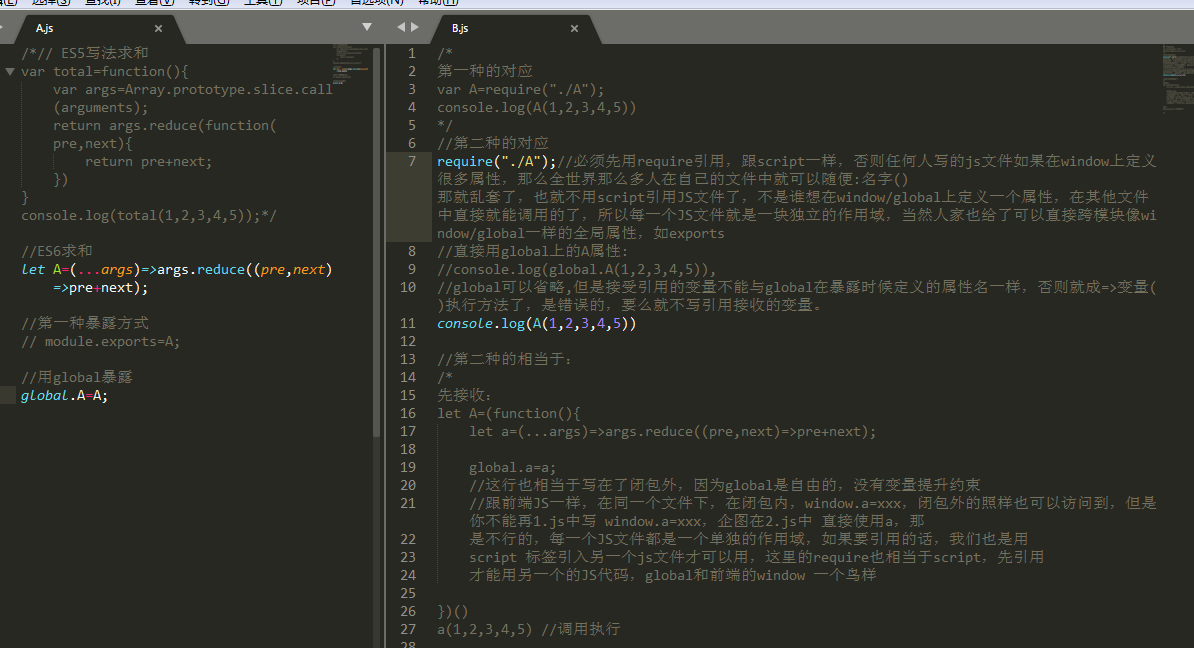
1-exports.xxx=Fn; =>给exports赋予属性可以导致module.exports对象变化，相当于exports.属性名=属性值放入了共同空间，而不是直接让exports=一个空间



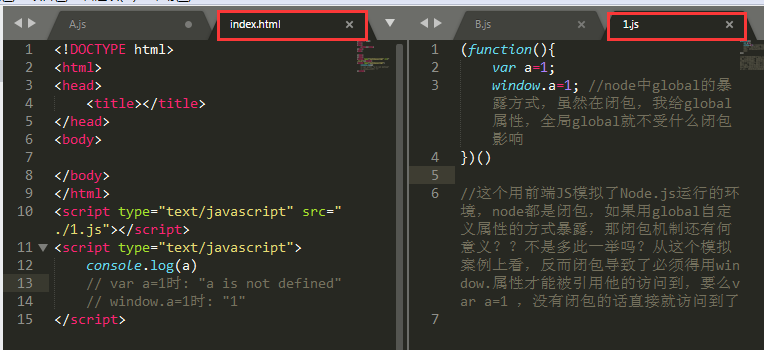
2-直接改变module.exports的指向，因为最终在看不见的执行闭包体内的原理就是return module.exports。

结论：要引用一个函数对象(空间)，用module.exports方便。

还可以利用global的全局性来时间模块之间的引用：



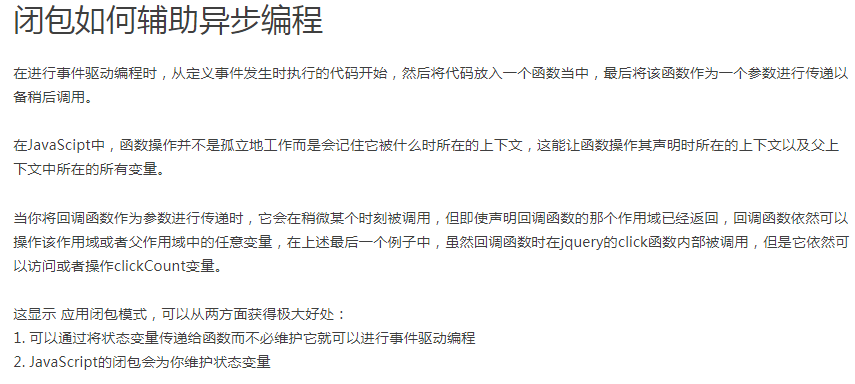
但是利用在global上定义是没有意义的，因为node.js文件就是一个闭包，看下图用前端JS模拟了node的闭包机制：



那么node为什么要用闭包机制呢？

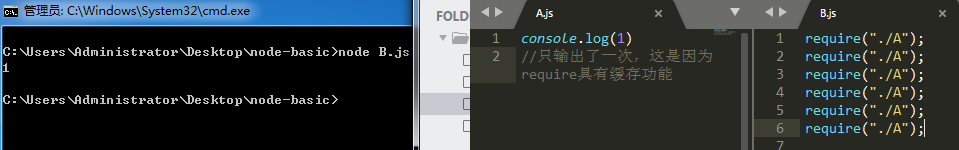
因为闭包可以形成一个私有作用域，node就是一个I/O，事件环，异步，回调特性...







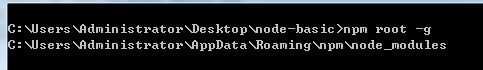
require具有缓存功能



**Npm:**

-第三方模块要通过npm进行安装 node pacakage manager

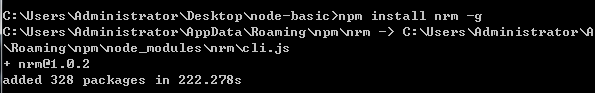
-全局安装 -g (只能在命令行(cmd)中使用) 默认的安装路径是(通过 npm root -g 查看)



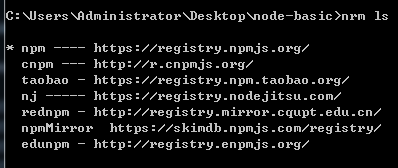
nrm:npm的源，里面有许多中国开发的(复制)的npm，因为npm是外国的，装东西很慢，用中国复制的会快很多，其实就是切换下载源头

安装命令：npm install nrm -g

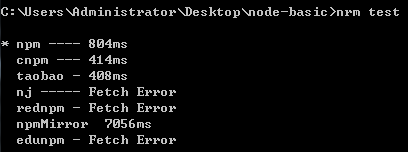
安装成功的图：



输入nrm ls =>查看nrm所有可用源：



nrm test =>查看各源链接时间



nrm use 源的名字 =>切换源



为什么能在npm中直接用nrm:因为npm是环境变量，nrm装在了npm里，npm带他飞~（通过npm映射）

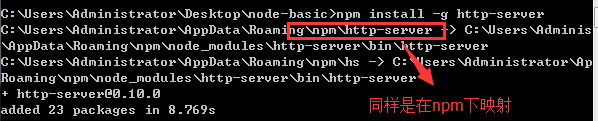


怎么卸载nrm?

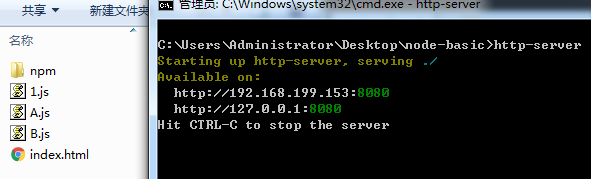
npm uninstall nrm -g

启动一个本地服务：（只能在命令行里用）

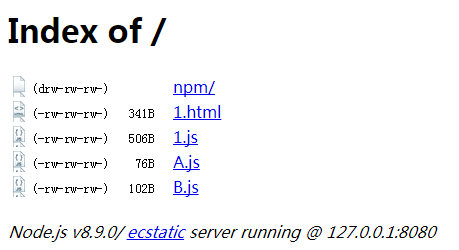
npm install -g http-server =>安装启动局域网和本地路径



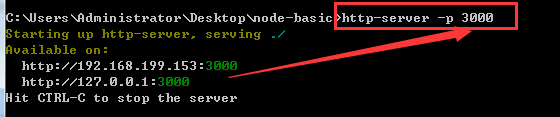
从要预览的项目文件夹中启动服务，直接输入http-server，可得到一个局域网ip和本机器IP地址，这样就可以访问了，也可以用手机去访问看看手机端的效果哈~



如果项目中没有index.html文件而是其他名字：则会出现像是阿帕奇一样的根目录地址：



如果不想再8080端口下启动，要修改端口：输入=> http-server -p 端口号



Idoc =>简单的文档生成工具

npm install idoc -g

具体各类玩意看npm官网上的说明

nvm:切换node版本的工具

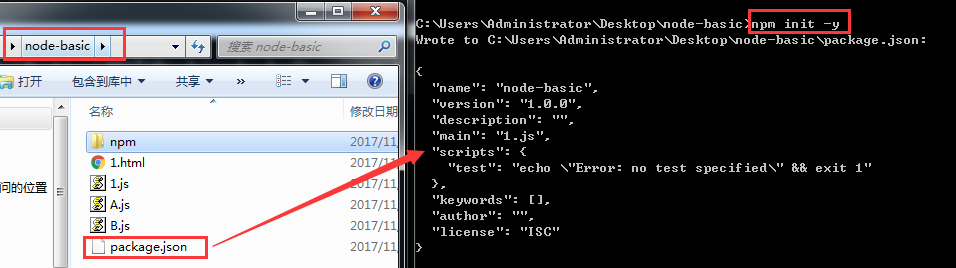
-**本地安装**

-没有-g参数，安装之前需要初始化，这个初始化文件是记录安装依赖的，比如说一个项目用了好多库和框架，那怎么知道用了哪些东西？就是靠这个Package.json文件。

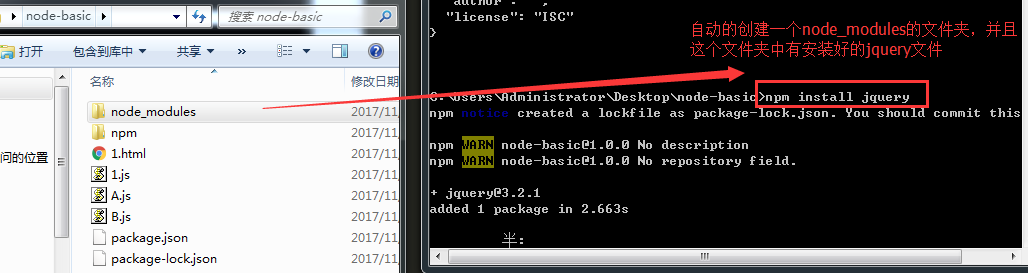
npm init -y =>默认初始化这个文件，生成 package.json,目录中不能有中文、特殊字符、大写，默认的包名和目录名相同，-y的意思就是默认的，如果不加-y 得一开头自己写一下。**默认先找当前目录下的package.json,如果当前没有会去上级查找，找不到才认为在当前目录下安装**

Package.json中scripts可以配置一些快捷方式，下面看图说明

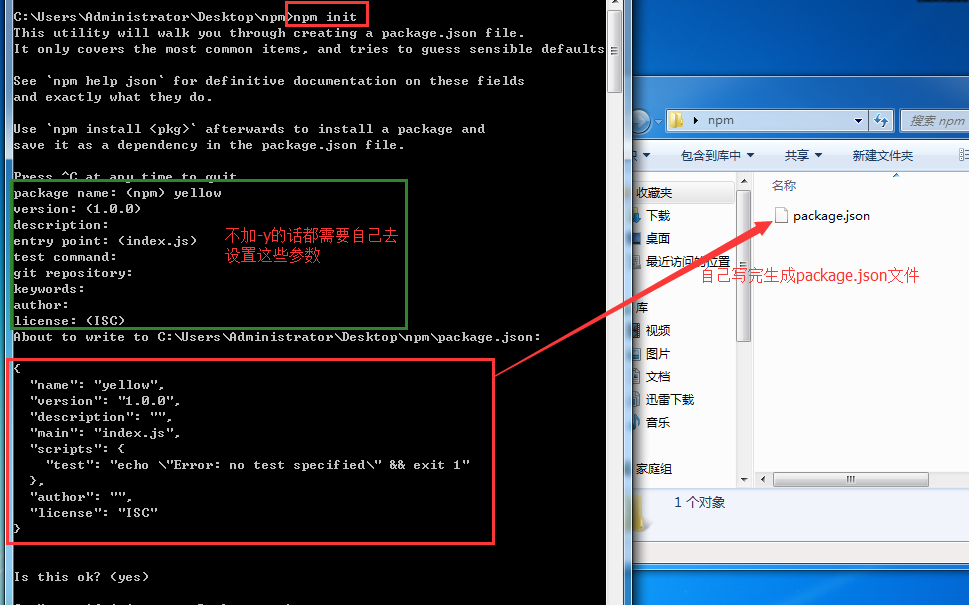
npm init -y 初始化，会默认在运行的项目文件夹中生成package.json文件



安装Jquery

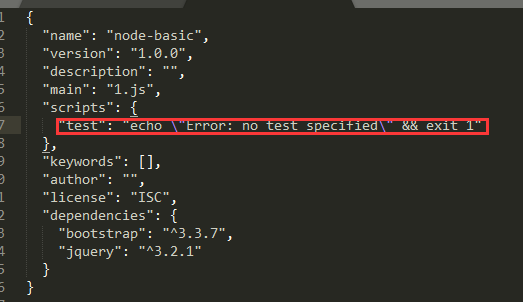


如果不加-y :



设置快捷方式：修改scripts中的内容

（原始的文件）



修改，设置快捷内容，以及运行时候输入=> npm run test



###项目依赖

-开发时使用，上线还需要

安装方式：npm install jquery (以前的Node版本还需要在后面加 --save)

指定版本安装：在要安装的东西后面+@版本号。如：npm install jquery@1.8.3

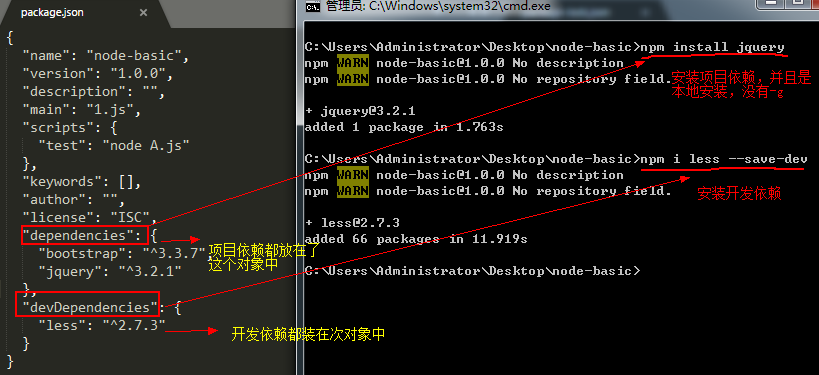
卸载方式：npm uninstall jquery

##开发依赖

-开发时使用，上线不使用

安装方式：npm install less --save-dev

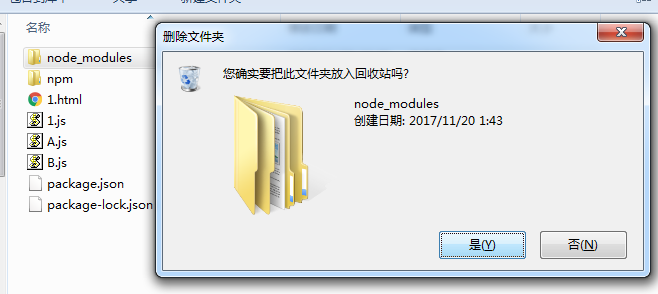
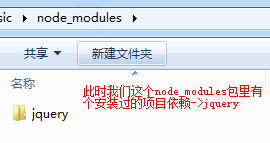
卸载方式：npm uninstall less --save-dev



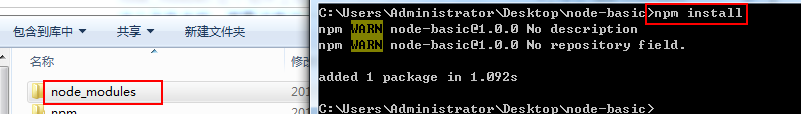
###安装全部依赖:

命令：npm install

我们开发的时候，里面有项目依赖如下面的jquery，上传到gitHub上的时候不可能把node\_modules这个包传上去，所以我们要把项目down下来的时候也是没有node\_modules这个文件夹的，那要没有这个node\_modules这个包里面所依赖的库，我们down下来怎么查看文件呢？这时候我们先把本地的node\_modules这个文件夹删除(模拟要上传的动作或者是从gitHub上下载下来就这样没有node\_modules这个包)，



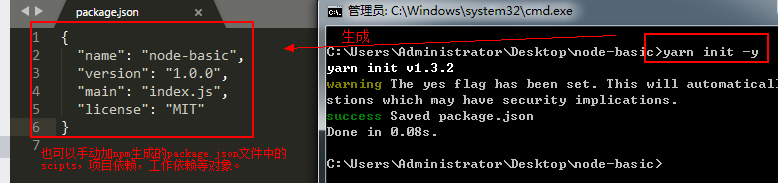
在命令行中输入npm install，会发现这个node\_modules又回来了，这是因为package.json这个文件之前已经记录了安装依赖文件（包括项目依赖和开发依赖时需要的）



###yarn安装 (不能像npm一样安装全局的，-g那种)

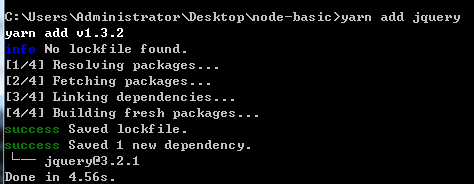
-安装：npm install -g yarn (安装后就不需要npm的package.json和node\_modules这些东西了，跟npm没关系了，他就是一个单独的包管理器)

第一步：初始化 yarn init -y (如果不加-y，还是要设置跟npm不加y时候的那些参数，加了也是类似npm加y已经默认的那些参数)

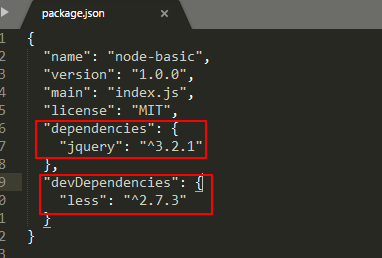
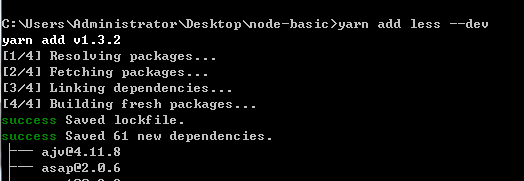


安装文件：

--项目依赖 yarn add jquery =>同样会生成node\_modules文件



--开发依赖： yarn add less --dev 生成的package.json文件



卸载：

项目依赖卸载：yarn remove jquery

开发依赖卸载：yan remove less --dev

###安装全部依赖：

yarn install =>跟npm的一个样

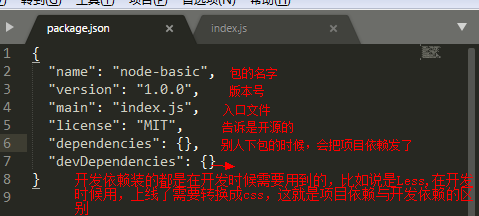
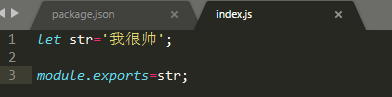
###怎么发布包：

-想要发包，首先要回到npm官网 =>nrm use npm (切换npm的地址)

-包名不能和已有的包一致 （是指与npm官网上已有的包名）

-入口文件，整合用 （index.js）

-注册npm账号 =>可通过命令行的方式注册 npm addUser,如果有账号，代表的是登陆。 （新用户需要校验邮箱，并且输入的时候密码是看不见光标的）

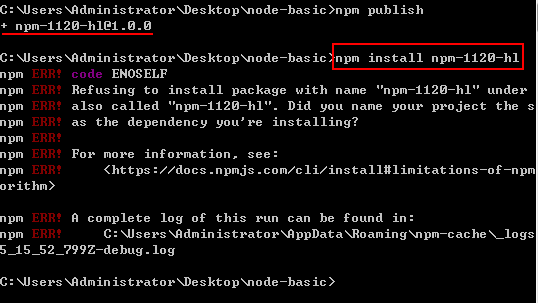
 

发布：npm publish (发布了名为的包)

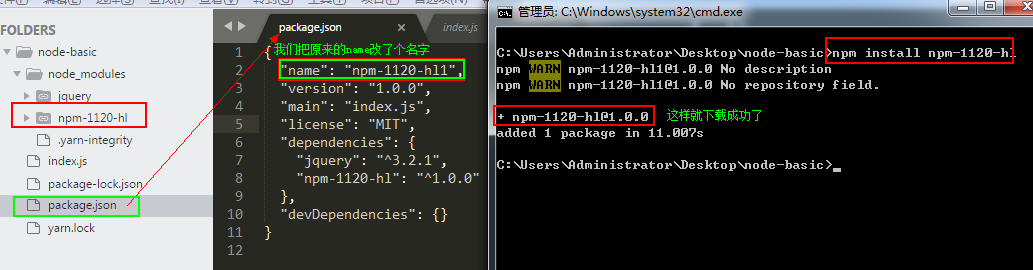


把发布的包如何下载下来？

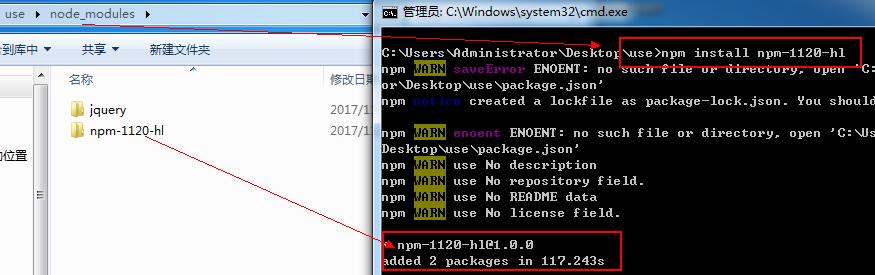
-注意下载的包名不能与要下载到目录名下的package.json中的name名相同的目录下。（就是包名相同）否则会报错。如下，我们把发布的包下载到了他自己的文件路径下了，报错了



意思就是不能自己安自己，我们把原来package.json中的name变了个名字，就可以安装了



我们新建了一个use的文件夹，并安装，（安装之前先要初始化一下npm init -y，防止安装到上一级，目录）



因为从npm的网站上安装要慢，我们切换到了taobao=>nrm use taobao (tabao这些站点会每隔10分钟会从npm官网更新一次)，图例省略，速度很快就是了。

###如何引包？

被引入的包如果直接写包名，那么这个包中被引用的JS名字必须是package.json中的main 所设置的入口文件名一致（这句其实是废话，因为发布包的时候入口文件名就是根据写的js文件名来设置的，这就明白为什么可以直接引包名）

