# 模块化

## 1

***console***.log(**"我是一个模块,我是02.module.js"**);  
  
  
*/\*  
我们可以通过 exports 来向外部暴露变量和方法  
 只需要将需要暴露给外部的变量或方法设置为exports的属性即可  
  
\* \*/  
//向外部暴露属性或方法****exports***.**x** = **"我是02.module.js中的x"**;  
***exports***.**y** = **"我是y"**;  
***exports***.fn = **function** () {  
  
};

## 2

*//引入其他的模块  
/\*  
 在node中，通过require()函数来引入外部的模块  
 require()可以传递一个文件的路径作为参数，node将会自动根据该路径来引入外部模块  
 这里路径，如果使用相对路径，必须以.或..开头  
  
 使用require()引入模块以后，该函数会返回一个对象，这个对象代表的是引入的模块  
  
 我们使用require()引入外部模块时，使用的就是模块标识，我们可以通过模块标识来找到指定的模块  
 - 模块分成两大类  
 核心模块  
 - 由node引擎提供的模块  
 - 核心模块的标识就是，模块的名字  
 文件模块  
 - 由用户自己创建的模块  
 - 文件模块的标识就是文件的路径（绝对路径，相对路径）  
 相对路径使用.或..开头  
  
 \*/  
//var md = require("./02.module");***var *math*** = *require*(**"./math"**);  
**var *fs*** = *require*(**"fs"**);  
  
*//console.log(md);****console***.log(***math***.**add**(123,456));  
*//console.log(fs);*

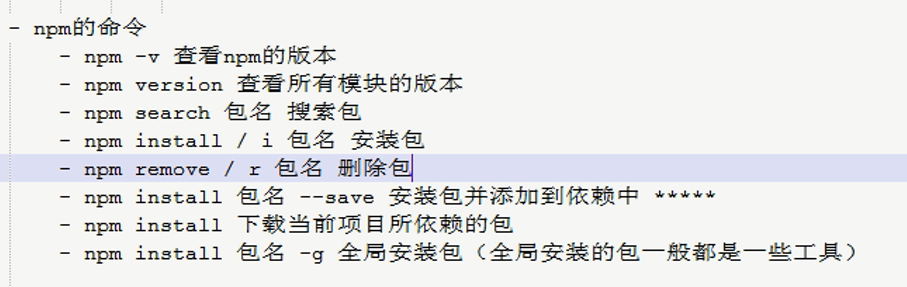
## 3

**var *a*** = 10;

// 在此，由于内部代码相当于包在一个函数里，所以这里的a 其实是一个局部变量，要想成为全局变量，应该直接:

a = 10;  
  
*/\*  
 在node中有一个全局对象 global，它的作用和网页中window类似  
 在全局中创建的变量都会作为global的属性保存  
 在全局中创建的函数都会作为global的方法保存  
  
 当node在执行模块中的代码时，它会首先在代码的最顶部，添加如下代码  
 function (exports, require, module, \_\_filename, \_\_dirname) {  
  
 在代码的最底部，添加如下代码  
 }  
  
 实际上模块中的代码都是包装在一个函数中执行的，并且在函数执行时，同时传递进了5个实参  
 exports  
 - 该对象用来将变量或函数暴露到外部  
  
 require  
 - 函数，用来引入外部的模块  
  
 module  
 - module代表的是当前模块本身  
 - exports就是module的属性  
 - 既可以使用 exports 导出，也可以使用module.exports导出  
  
 \_\_filename  
 C:\Users\lilichao\WebstormProjects\class0705\01.node\04.module.js  
 - 当前模块的完整路径  
  
 \_\_dirname  
 C:\Users\lilichao\WebstormProjects\class0705\01.node  
 - 当前模块所在文件夹的完整路径  
  
\* \*/  
//console.log(global.a);  
  
/\*  
 arguments.callee  
 - 这个属性保存的是当前执行的函数对象  
\* \*/  
//console.log(arguments.callee + "");  
//console.log(arguments.length);  
  
//console.log(exports);  
//console.log(module.exports == exports);****console***.log(***\_\_dirname***);

## npm



# fs(文件系统)

## buffer

*/\*  
 Buffer(缓冲区)  
 - Buffer的结构和数组很像，操作的方法也和数组类似  
 - 数组中不能存储二进制的文件，而buffer就是专门用来存储二进制数据  
 - 使用buffer不需要引入模块，直接使用即可  
 - 在buffer中存储的都是二进制数据，但是在显示时都是以16进制的形式显示  
 buffer中每一个元素的范围是从00 - ff 0 - 255  
 00000000 - 11111111  
  
 计算机 一个0 或一个1 我们称为1位（bit）  
  
 8bit = 1byte（字节）  
 1024byte = 1kb  
 1024kb = 1mb  
 1024mb = 1gb  
 1024gb = 1tb  
  
 buffer中的一个元素，占用内存的一个字节  
  
 - Buffer的大小一旦确定，则不能修改，Buffer实际上是对底层内存的直接操作  
  
 \*/***var *str*** = **"Hello 尚硅谷"**;  
  
*//将一个字符串保存到buffer中***var *buf*** = ***Buffer***.from(***str***);  
  
*//console.log(buf.length); //占用内存的大小  
//console.log(str.length);//字符串的长度  
//console.log(buf);  
  
//创建一个指定大小的buffer  
//buffer构造函数都是不推荐使用的  
//var buf2 = new Buffer(10);//10个字节的buffer  
//console.log(buf2.length);  
  
//创建一个10个字节的buffer***var *buf2*** = ***Buffer***.alloc(10);  
*//通过索引，来操作buf中的元素****buf2***[0] = 88;  
***buf2***[1] = 255;  
***buf2***[2] = 0xaa;  
***buf2***[3] = 255;  
  
*//只要数字在控制台或页面中输出一定是10进制  
//console.log(buf2[2].toString(16));  
  
/\*for(var i=0 ; i<buf2.length ; i++){  
 console.log(buf2[i]);  
}\*/  
  
//Buffer.allocUnsafe(size) 创建一个指定大小的buffer，但是buffer中可能含有敏感数据  
/\*var buf3 = Buffer.allocUnsafe(10);  
console.log(buf3);\*/  
  
/\*  
 Buffer.from(str) 将一个字符串转换为buffer  
 Buffer.alloc(size) 创建一个指定大小的Buffer  
 Buffer.alloUnsafe(size) 创建一个指定大小的Buffer，但是可能包含敏感数据  
 buf.toString() 将缓冲区中的数据转换为字符串  
 \*/***var *buf4*** = ***Buffer***.from(**"我是一段文本数据"**);  
  
***console***.log(***buf4***.toString());

## 同步文件写入

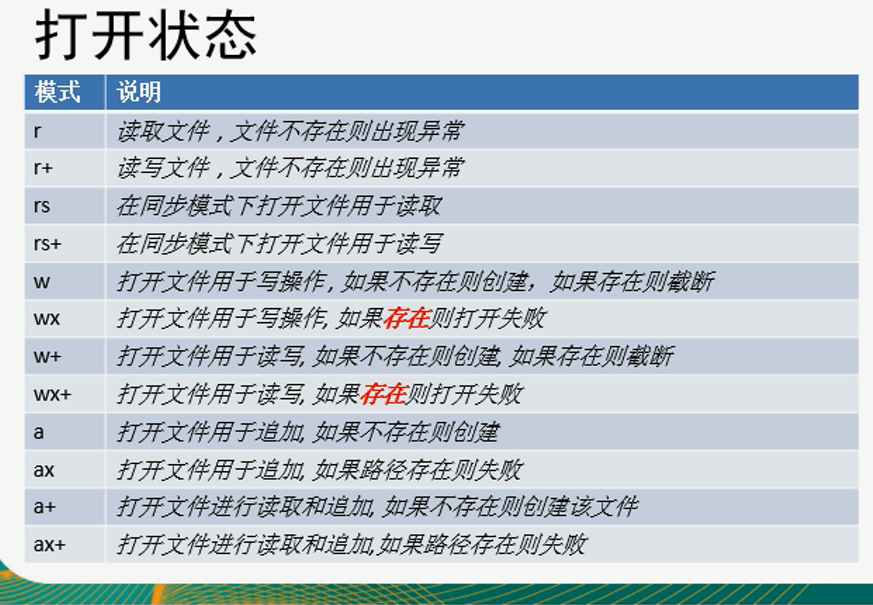
*/\*  
 文件系统（File System）  
 - 文件系统简单来说就是通过Node来操作系统中的文件  
 - 使用文件系统，需要先引入fs模块，fs是核心模块，直接引入不需要下载  
  
 同步文件的写入  
 - 手动操作的步骤  
 1.打开文件  
 fs.openSync(path, flags[, mode])  
 - path 要打开文件的路径  
 - flags 打开文件要做的操作的类型  
 r 只读的  
 w 可写的  
 - mode 设置文件的操作权限，一般不传  
 返回值：  
 - 该方法会返回一个文件的描述符作为结果，我们可以通过该描述符来对文件进行各种操作  
  
 2.向文件中写入内容  
 fs.writeSync(fd, string[, position[, encoding]])  
 - fd 文件的描述符，需要传递要写入的文件的描述符  
 - string 要写入的内容  
 - position 写入的起始位置  
 - encoding 写入的编码，默认utf-8  
  
 3.保存并关闭文件  
 fs.closeSync(fd)  
 - fd 要关闭的文件的描述符  
  
  
  
 \*/***var *fs*** = *require*(**"fs"**);  
  
*//打开文件***var *fd*** = ***fs***.openSync(**"hello.txt"** , **"w"**);  
  
*//向文件中写入内容****fs***.writeSync(***fd*** , **"今天天气真不错~~~"**, 2);  
  
*//关闭文件****fs***.closeSync(***fd***);  
  
***console***.log(**"程序向下执行~~~"**);

## 异步文件写入

*/\*\*  
 异步文件写入  
 fs.open(path, flags[, mode], callback)  
 - 用来打开一个文件  
 - 异步调用的方法，结果都是通过回调函数的参数返回的  
 - 回调函数两个参数：  
 err 错误对象，如果没有错误则为null  
 fd 文件的描述符  
 fs.write(fd, string[, position[, encoding]], callback)  
 - 用来异步写入一个文件  
  
 fs.close(fd, callback)  
 - 用来关闭文件  
  
 \*/  
  
//引入fs模块***var *fs*** = *require*(**"fs"**);  
  
  
*//打开文件****fs***.open(**"hello2.txt"**,**"w"**,**function** (err , fd) {  
 *//判断是否出错* **if**(!err){  
 *//如果没有出错，则对文件进行写入操作* ***fs***.write(fd,**"这是异步写入的内容"**,**function** (err) {  
 **if**(!err){  
 ***console***.log(**"写入成功~~"**);  
 }  
 *//关闭文件* ***fs***.close(fd , **function** (err) {  
 **if**(!err){  
 ***console***.log(**"文件已关闭~~~"**);  
 }  
 });  
 });  
 }**else**{  
 ***console***.log(err);  
 }  
});  
  
***console***.log(**"程序向下执行~~~"**);

## 简单文件写入

*/\*  
 简单文件写入  
 fs.writeFile(file, data[, options], callback)  
 fs.writeFileSync(file, data[, options])  
 - file 要操作的文件的路径  
 - data 要写入的数据  
 - options 选项，可以对写入进行一些设置  
 - callback 当写入完成以后执行的函数  
  
 - flag  
 r 只读  
 w 可写  
 a 追加  
 \*/  
//引入fs模块***var *fs*** = *require*(**"fs"**);  
  
*/\*fs.writeFile("hello3.txt","这是通过writeFile写入的内容",{flag:"r+"} , function (err) {  
 if(!err){  
 console.log("写入成功~~~");  
 }else{  
 console.log(err);  
 }  
});\*/  
  
  
//C:\Users\lilichao\Desktop\hello.txt  
//C:\\Users\\lilichao\\Desktop\\hello.txt****fs***.writeFile(**"C:/Users/lilichao/Desktop/hello.txt"**,**"这是通过writeFile写入的内容"**,{**flag**:**"w"**} , **function** (err) {  
 **if**(!err){  
 ***console***.log(**"写入成功~~~"**);  
 }**else**{  
 ***console***.log(err);  
 }  
});



## 流式文件写入

*/\*  
 同步、异步、简单文件的写入都不适合大文件的写入，性能较差，容易导致内存溢出  
 \*/***var *fs*** = *require*(**"fs"**);  
  
*//流式文件写入  
//创建一个可写流  
/\*  
 fs.createWriteStream(path[, options])  
 - 可以用来创建一个可写流  
 - path，文件路径  
 - options 配置的参数  
 \*/***var *ws*** = ***fs***.createWriteStream(**"hello3.txt"**);  
  
*//可以通过监听流的open和close事件来监听流的打开和关闭  
/\*  
 on(事件字符串,回调函数)  
 - 可以为对象绑定一个事件  
  
 once(事件字符串,回调函数)  
 - 可以为对象绑定一个一次性的事件，该事件将会在触发一次以后自动失效  
  
\* \*/****ws***.once(**"open"**,**function** () {  
 ***console***.log(**"流打开了~~~"**);  
});  
  
***ws***.once(**"close"**,**function** () {  
 ***console***.log(**"流关闭了~~~"**);  
});  
  
*//通过ws向文件中输出内容****ws***.write(**"通过可写流写入文件的内容"**);  
***ws***.write(**"今天天气真不错"**);  
***ws***.write(**"锄禾日当午"**);  
***ws***.write(**"红掌拨清清"**);  
***ws***.write(**"清清真漂亮"**);  
  
*//关闭流****ws***.end();