今天我们来学习一下nodejs的模型构成作为学习nodejs的开篇，希望我能够将nodejs学习通透，然后慢慢在项目中进行体验开发。根据nodejs文档的描述，在nodejs中每一个文件就是一个模型，而引用某个模型将使用他只有的API来进行载入，同时将有关键字用于公开（如.NET中的public）变量或方法。

**载入模型（require）：**

要使用模型首先需要载入模型，在nodejs中使用require关键字来进行载入，相当于.NET中的using，Java中的import等等。加入两个文件在同一个目录开使用如下格式进行载入：

var circle = require('./circle.js');

console.log( 'The area of a circle of radius 4 is '

+ circle.area(4));

其中circle.js所处的位置和当前js文件所在的文件夹是同一个，下面我们来看一下导出关键字。

**导出（exports）：**

在每个模型中使用var直接声明的变量为私有变量，只能在当前模型中使用，如果在其他模型中要使用当前变量或方法就需要使用exports。这个相当于客户端脚本中的window，我在使用jQuery开发模式时候，要向文档公开当前变量，就会使用到window。

var PI = Math.PI;

exports.area = function (r) {

return PI \* r \* r;

};

exports.circumference = function (r) {

return 2 \* PI \* r;

};

在上面这个代码中PI就是私有变量，在模型外部是不能使用的，而area和circumference两个函数在外部是可以被使用的。

**相互引用（cycles）：**

相信写代码的同志们就会发现，如果依上面两个的使用方法，那么如a.js引用b.js，而b.js又载入a.js，这样不是会照成死循环。不过nodejs团队为我们解决了这个问题，如果a.js模型先载入b.js模型，那样和前面使用情况一样，如果在b.js中又载入a.js，那样a.js模型将直接公开a.js模型中所有变量或方法，而不会执行a.js中的代码。我们可以先看个例子，那样比较好理解：

//a.js

console.log('a starting');

exports.done = false;

var b = require('./b.js');

console.log('in a, b.done = %j', b.done);

exports.done = true;

console.log('a done');

//b.js

console.log('b starting');

exports.done = false;

var a = require('./a.js');

console.log('in b, a.done = %j', a.done);

exports.done = true;

console.log('b done');

//main.js

console.log('main starting');

var a = require('./a.js');

var b = require('./b.js');

console.log('in main, a.done=%j, b.done=%j', a.done, b.done);

我们执行main.js就会得到如下结果，从结果中就可以很容易发现他执行的顺序，而那个先引用先自行，都是看调用的先后，因为js是属于解析性语言。

$ node main.js

main starting

a starting

b starting

in b, a.done = false

b done

in a, b.done = true

a done

in main, a.done=true, b.done=true

**核心模型（core modules）：**

核心模型被放在Nodejs安装目录下的**lib/**目录，核心模型使用ID来进行引用，例如require('HTTP')，将引入HTTP模型。核心模型具有最高的优先级，如：require('HTTP')，即使存在http.js文件也不会被引入，而引入的是核心模块中的HTTP模型。

**文件模型：**

在载入文件模型时，假如文件扩展名不存在，nodejs将试图附加.js，.json，.node进行查找。其中：

1. .js将像通用的模型被载入，
2. .json扩展名的文件模型，将被解析为一个JSON对象，
3. .node将使用dlopen引入扩展的已编译的插件。

三个扩展名具有三个不同的格式，在引入文件模型的时候，不同路径也将被解析为不一样的模型：

1. “/”使用这个字符开头的路径将解析为绝对路径，这个和WEB中使用一样；
2. "./"使用这个字符串开头的路径，两个模型必须处于同一个目录中；
3. “../”相对目录中；
4. 如果都不是，将从核心模型中获取，或nodejs的**模型文件夹（node\_modules）**中进行查找。

如果上面三个都没找到，将抛出错误：MODULE\_NOT\_FOUND。

**从模型文件夹中进行载入文件：**

在当前路径中载入文件模型，如果当前路径找不到，将会自动在父目录树中进行查找，例如：在'/home/ry/projects/foo.js'中使用require('bar.js')，将尝试引用的优先级别如下：

* /home/ry/projects/node\_modules/bar.js
* /home/ry/node\_modules/bar.js
* /home/node\_modules/bar.js
* /node\_modules/bar.js

**从模型文件夹中进行载入文件夹：**

在nodejs中，也可以和.net的一样的做法，在目录中创建一个package.json文件，然后在里面进行配置，具体如下：

{ "name" : "some-library",

"main" : "./lib/some-library.js" }

假设这个文件在目录somelibs下，在使用require('./somelibs')时，将载入：./somelibs/lib/some-library.js的模型。如果不存在这个package.json文件时，引用当前文件夹如:require('./somelibs')，将会尝试引入如下文件：

1. ./somelibs/index.js
2. ./somelibs/index.json

**缓存模型：**

上一篇中我们知道了如何引入模型，当模型被引用后将被nodejs系统缓存，所以在后面引用只会直接从缓存中调用。这是一个很有用的特性，因为不会重复解析，并且可以得到相同的对象。而缓存这些对象信息的以依靠其文件路径来作为唯一键的，所以只要是同一个文件内的模型就会得到相同的对象。

*注意：如果使用引入的路径会照成多个解析，有可能引入的是不同对象，例如上一篇中讲到的引入解析顺序，相同的键可能引入不同的对象。*

参考文献：

<http://www.mozlite.com/nodejs-modules>