缺点：

无法利用多核CPU

错误会引起整个应用退出（应用的健壮性）

大量计算占用CPU导致无法继续调用异步 I/O

由于单线程，大量计算（耗时），将会导致CPU时间片不能释放，使得后续I/O无法发起。

# NODE命令行

node --version

node进入 REPL,按两次ctrl + c退出。

# Node的模块实现

核心模块，文件模块，自定义模块

核心模块（node提供的模块，其加载优先级仅仅次于缓存的优先级。高于文件模块）

1. 路径分析

模块标识符分析

require()方法接受一个标识符作为参数

模块标识符分为：

核心模块 如http, fs , path

. 或 ..开始的相对路径

以 /开始的相对路径文件模块

非路径形式的文件模块，如自定义的connect模块（？？？）

路径形式加载的文件，会将路径转换为真实路径，并以真实路径为索引，将编译后的执行结果存放在缓存中。

自定义模块（node\_module）

会从当前文件目录中逐级向上递归查找 node\_module文件，文件层级越深，模块查询耗时越多（自定义模块加载最慢）

1. 文件定位

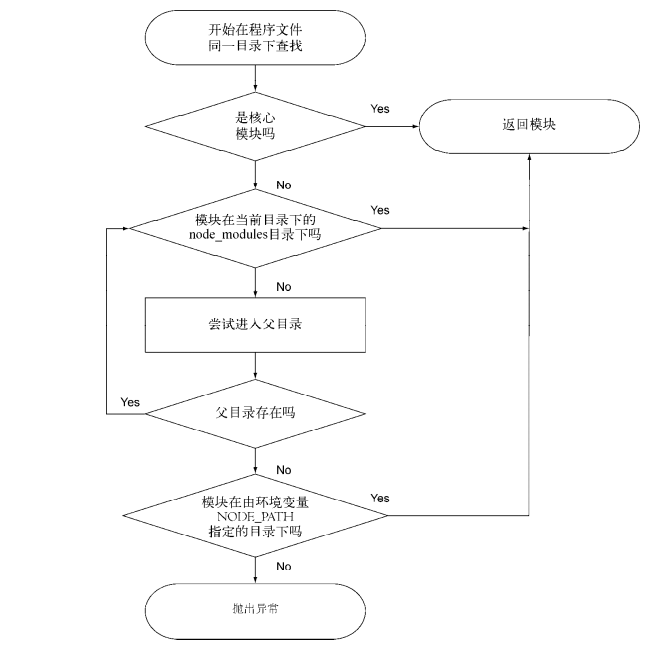
需要注意的是

文件扩展名分析

由于commen.js允许标识符不带扩展名，在没有写扩展名时，node会按照.js , .json, .node的次序补足扩展名。而尝试时，会调用fs模块阻塞式判断文件是否存在（单线程会引发性能问题）

因而在 .node .json文件时，应该带上标识符

目录分析和包



1. 编译分析

定位到具体文件后，node会新建一个模块对象，然后根据路径载入并编译。对于不同的文件扩展名，其载入方式有所不同

.js 通过fs模块同步读取文件后编译执行

.node 这是用 c/c++ 编写的扩展文件，通过 dlopen()方法加载最后编生成的文件。

.json 通过fs模块同步读取，用JSON.parse()解析返回结果

其他扩展名的文件（都会被当做js文件载入）

注意：部分核心模块在node编译过程中，编译进了二进制执行文件中，在node进程启动时，部分核心模块就被直接加载进了内存。（因而这部分核心模块引入时，文件定位和编译分析这两部可以省略）

优先从缓存中加载（二次加载）

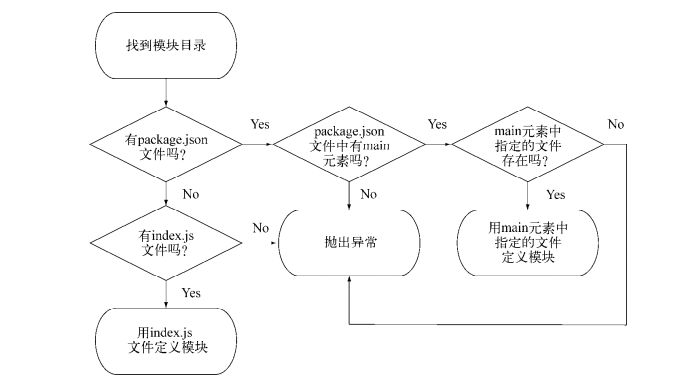
node会对已经引入过的模块进行缓存，与浏览器缓存不同的是，node缓存的是编译和执行后的对象。

注意：

exports 、 module.export与 export 、export default的联系和区别

**t\_id：export\_1516687482071**

如果模块是一个目录，则模块目录中必须有一个被命名为index.js的文件（除非在这个目录下的package.json中特别指定，main:’path’）



# 核心模块

## fs :文件模块

异步

fs.readFile(url, ‘utf-8’, function(err, data){})

同步

fs.readFileSync(url,”utf-8”, function(err, data){})

打开文件

fs.open(path,flags,[mode], [cb])

## util: 常用工具模块

提供常用函数的集合，用于弥补JavaScript功能的不足。

util.inherits

实现了原型继承，但是实际上

util.inspect

将任意一个对象转换为字符串的方法。

## event： 事件模块

## http: 封装了服务器和一个简易的客户端

http.Server()是一个基于事件的http服务器，所有请求都被封装为独立的事件，开发者只需要对它的事件编写响应函数即可实现http服务器所有的功能。（继承自EventEmitter）

var server = new http.Server()

server事件

request（）

connection : keep-alive时可能会在同一个连接内发送多次请求

close： 服务器关闭时，事件触发。（不是用户断开连接）

http.createServer(function(req,res){

}).listen(8080)

实际上等于

var server = new http.Server();

server.on(“request”, function(req, res){

})

server.listen(8080)

接收post数据

post请求的内容都在请求体中，req对象中

## Url: url模块

url.parse(url, parseQueryString, slashesDenoteHost)

返回一个解析后的对象

## 事件

异步操作完成后都会发送一个事件到事件队列。对于开发者，由EventEmitter对象提供的

# 异步I/O

## 操作系统的异步I/O

操作系统非阻塞调用，会立即返回（不带数据，而是通过文件描述符管理）

然后通过轮询，查询数据。

轮询技术

read

select

poll

epoll

linux下效率最高的i/o事件通知机制，在轮询时，没有检查到i/o事件，将进行休眠，知道事件将其唤醒。（事件通知，执行回调的方式）

## node的异步I/O

而node实现了线程池完成异步 I/O, 所以说平常所得node是单线程的，实际上指定是JavaScript执行在单线程中罢了，在node ，无论是\*nix 还是 windows平台，内部完成I/O任务的另有线程池。

## 事件循环

在进程启动时，node便会创建类似while的循环。没执行一次循环体的过程我们成为Tick。事件循环是一个典型的生产者/消费者模型。异步I/O，网络请求则是事件的生产者，源源不断为Node提供不同类型的事件，这些事件被传递到对应的观察者。事件循环从观察者哪里取出事件并处理。

# 全局对象与全局变量

node中定义一个全局变量，这个变量也会成为全局对象的属性，反之亦然。

nodejs中全局对象是global

内置的全局属性

process

描述nodejs进程状态的对象

process.argv : 命令行参数数组

process.stdout: 标准输入流

process.stdin: 标准输入流

process.nextTick(cb) :为下次事件循环设置任务，node.js会在下次事件循环时调用cb

这个目的主要是针对计算密集的计算的拆分（密集型计算占用大量CPU时间），因此nodejs原则是尽量缩短每个事件的执行时间。

process.platform process.pid process.execPath process.memoryUsage()

等等