目录

[1、Nodejs异常处理](#_Toc9610_WPSOffice_Level1) [3](#_Toc9610_WPSOffice_Level1)

[2、assert模块](#_Toc12141_WPSOffice_Level1) [3](#_Toc12141_WPSOffice_Level1)

[3、Buffer](#_Toc276_WPSOffice_Level1) [4](#_Toc276_WPSOffice_Level1)

[（1）、ascii](#_Toc12141_WPSOffice_Level2) [4](#_Toc12141_WPSOffice_Level2)

[（2）、utf-8](#_Toc276_WPSOffice_Level2) [4](#_Toc276_WPSOffice_Level2)

[（3）、base64](#_Toc9190_WPSOffice_Level2) [4](#_Toc9190_WPSOffice_Level2)

[（4）、utf16le：UTF-16的小端编码，支持大于U+10000的四字节字符](#_Toc17808_WPSOffice_Level2) [4](#_Toc17808_WPSOffice_Level2)

[（5）、hex：将每个字节转为两个十六进制字符](#_Toc3557_WPSOffice_Level2) [4](#_Toc3557_WPSOffice_Level2)

[（1）、write()](#_Toc2526_WPSOffice_Level2) [4](#_Toc2526_WPSOffice_Level2)

[（2）、slice()](#_Toc16439_WPSOffice_Level2) [4](#_Toc16439_WPSOffice_Level2)

[（3）、toString()](#_Toc25289_WPSOffice_Level2) [5](#_Toc25289_WPSOffice_Level2)

[（4） toJSON()](#_Toc7405_WPSOffice_Level2) [5](#_Toc7405_WPSOffice_Level2)

[4、Child Process 模块](#_Toc9190_WPSOffice_Level1) [5](#_Toc9190_WPSOffice_Level1)

[1、exec()](#_Toc28902_WPSOffice_Level2) [5](#_Toc28902_WPSOffice_Level2)

[2、execFile()](#_Toc30207_WPSOffice_Level2) [5](#_Toc30207_WPSOffice_Level2)

[5、文件查找策略](#_Toc17808_WPSOffice_Level1) [5](#_Toc17808_WPSOffice_Level1)

[1、require()查找优先级的顺序：](#_Toc2843_WPSOffice_Level2) [5](#_Toc2843_WPSOffice_Level2)

[A. 返回该模块。](#_Toc12141_WPSOffice_Level3) [5](#_Toc12141_WPSOffice_Level3)

[B. 不再继续执行](#_Toc276_WPSOffice_Level3) [5](#_Toc276_WPSOffice_Level3)

[A. 根据X所在的父模块，确定X的绝对路径。](#_Toc9190_WPSOffice_Level3) [5](#_Toc9190_WPSOffice_Level3)

[B. 将X当成文件，一次查按照以下的顺序查找，只要存在即返回停止：](#_Toc17808_WPSOffice_Level3) [6](#_Toc17808_WPSOffice_Level3)

[1. X ,-->2. X.js ,--> 3. X.json -->4. X.node](#_Toc10305_WPSOffice_Level2) [6](#_Toc10305_WPSOffice_Level2)

[C. 将X当成目录，一次查找下面的文件，只要存在即返回停止：](#_Toc3557_WPSOffice_Level3) [6](#_Toc3557_WPSOffice_Level3)

[1、 X/package.json](#_Toc7755_WPSOffice_Level2) [6](#_Toc7755_WPSOffice_Level2)

[2、 X/index.js](#_Toc27370_WPSOffice_Level2) [6](#_Toc27370_WPSOffice_Level2)

[3、 X/index.json](#_Toc29512_WPSOffice_Level2) [6](#_Toc29512_WPSOffice_Level2)

[4、 X/index.node](#_Toc24618_WPSOffice_Level2) [6](#_Toc24618_WPSOffice_Level2)

[D. 如果X不带路径](#_Toc2526_WPSOffice_Level3) [6](#_Toc2526_WPSOffice_Level3)

[A. 根据X所在的父模块，确定X可能安装的目录。](#_Toc16439_WPSOffice_Level3) [6](#_Toc16439_WPSOffice_Level3)

[B. 依次在每个目录中，将X当成文件名或目录名加载。](#_Toc25289_WPSOffice_Level3) [6](#_Toc25289_WPSOffice_Level3)

[E. 抛出not found“”](#_Toc7405_WPSOffice_Level3) [6](#_Toc7405_WPSOffice_Level3)

[6、cluster模块](#_Toc3557_WPSOffice_Level1) [6](#_Toc3557_WPSOffice_Level1)

[1、 worker对象](#_Toc21891_WPSOffice_Level2) [6](#_Toc21891_WPSOffice_Level2)

[（1） 、worker.id](#_Toc28902_WPSOffice_Level3) [6](#_Toc28902_WPSOffice_Level3)

[（2） 、worker.process](#_Toc30207_WPSOffice_Level3) [6](#_Toc30207_WPSOffice_Level3)

[（3） 、worker.send()](#_Toc2843_WPSOffice_Level3) [7](#_Toc2843_WPSOffice_Level3)

[2、 Cluster.workers对象](#_Toc6059_WPSOffice_Level2) [7](#_Toc6059_WPSOffice_Level2)

[3、 Cluster模块的属性与方法](#_Toc10257_WPSOffice_Level2) [7](#_Toc10257_WPSOffice_Level2)

[（1） 、isMaster，isWorker](#_Toc10305_WPSOffice_Level3) [7](#_Toc10305_WPSOffice_Level3)

[（2） 、fork()](#_Toc7755_WPSOffice_Level3) [7](#_Toc7755_WPSOffice_Level3)

[（3） 、kill()](#_Toc27370_WPSOffice_Level3) [7](#_Toc27370_WPSOffice_Level3)

[（4） 、listening事件](#_Toc29512_WPSOffice_Level3) [7](#_Toc29512_WPSOffice_Level3)

[4、 不中断地重启Node服务](#_Toc31219_WPSOffice_Level2) [7](#_Toc31219_WPSOffice_Level2)

[首先，主进程向worker进程发出重启信号。](#_Toc24618_WPSOffice_Level3) [7](#_Toc24618_WPSOffice_Level3)

[5、 PM模块](#_Toc9212_WPSOffice_Level2) [7](#_Toc9212_WPSOffice_Level2)

[7、Node应用程序开发](#_Toc2526_WPSOffice_Level1) [8](#_Toc2526_WPSOffice_Level1)

[（1）、启动](#_Toc3074_WPSOffice_Level2) [8](#_Toc3074_WPSOffice_Level2)

[8、events模块](#_Toc16439_WPSOffice_Level1) [8](#_Toc16439_WPSOffice_Level1)

[（1） 、概述](#_Toc4603_WPSOffice_Level2) [8](#_Toc4603_WPSOffice_Level2)

[（2） 、Event Emitter 接口的部署](#_Toc13469_WPSOffice_Level2) [8](#_Toc13469_WPSOffice_Level2)

[（3） 、Event Emitter的实例方法](#_Toc8963_WPSOffice_Level2) [9](#_Toc8963_WPSOffice_Level2)

[（4） 、错误捕获](#_Toc14622_WPSOffice_Level2) [9](#_Toc14622_WPSOffice_Level2)

[（5） 、事件类型](#_Toc26195_WPSOffice_Level2) [9](#_Toc26195_WPSOffice_Level2)

[1）、newListener事件：添加新的回调函数时触发。](#_Toc22890_WPSOffice_Level3) [9](#_Toc22890_WPSOffice_Level3)

[2）、removeListener事件：移除回调时触发。](#_Toc18140_WPSOffice_Level3) [9](#_Toc18140_WPSOffice_Level3)

[9、Express框架](#_Toc22890_WPSOffice_Level1) [9](#_Toc22890_WPSOffice_Level1)

[1、 运行原理](#_Toc22890_WPSOffice_Level2) [9](#_Toc22890_WPSOffice_Level2)

[2、 中间件](#_Toc18140_WPSOffice_Level2) [10](#_Toc18140_WPSOffice_Level2)

[3、 use方法](#_Toc485_WPSOffice_Level2) [10](#_Toc485_WPSOffice_Level2)

[4、 Express的方法](#_Toc28122_WPSOffice_Level2) [10](#_Toc28122_WPSOffice_Level2)

[5、response对象](#_Toc7483_WPSOffice_Level2) [10](#_Toc7483_WPSOffice_Level2)

[（1）response.redirect方法](#_Toc485_WPSOffice_Level3) [10](#_Toc485_WPSOffice_Level3)

[（2）response.sendFile方法](#_Toc28122_WPSOffice_Level3) [10](#_Toc28122_WPSOffice_Level3)

[（3）response.render方法](#_Toc7483_WPSOffice_Level3) [10](#_Toc7483_WPSOffice_Level3)

[6、动态模板引擎](#_Toc5639_WPSOffice_Level2) [10](#_Toc5639_WPSOffice_Level2)

[1、设置静态文件的目录地址，指定一个文件为静态文件的目录：](#_Toc13512_WPSOffice_Level2) [10](#_Toc13512_WPSOffice_Level2)

[2、加载页面模板：使用render来加载渲染页面](#_Toc9033_WPSOffice_Level2) [10](#_Toc9033_WPSOffice_Level2)

[10、fs模块](#_Toc18140_WPSOffice_Level1) [10](#_Toc18140_WPSOffice_Level1)

[1、 readFile()、readFileSync()](#_Toc1257_WPSOffice_Level2) [11](#_Toc1257_WPSOffice_Level2)

[2、 writeFile()，writeFileSync()](#_Toc28754_WPSOffice_Level2) [11](#_Toc28754_WPSOffice_Level2)

[1)、writeFile方法用于异步写入文件。](#_Toc5639_WPSOffice_Level3) [11](#_Toc5639_WPSOffice_Level3)

[2)、writeFileSync方法用于同步写入文件。](#_Toc13512_WPSOffice_Level3) [11](#_Toc13512_WPSOffice_Level3)

[3、exits(path,callback)](#_Toc30251_WPSOffice_Level2) [11](#_Toc30251_WPSOffice_Level2)

[4、mkdir()，writeFile()，readFile()](#_Toc24982_WPSOffice_Level2) [12](#_Toc24982_WPSOffice_Level2)

[1)、mkdir方法用于新建目录。](#_Toc9033_WPSOffice_Level3) [12](#_Toc9033_WPSOffice_Level3)

[2) 、writeFile方法用于写入文件。](#_Toc1257_WPSOffice_Level3) [12](#_Toc1257_WPSOffice_Level3)

[3)、readFile方法用于读取文件内容。](#_Toc28754_WPSOffice_Level3) [12](#_Toc28754_WPSOffice_Level3)

[5、 stat()](#_Toc26852_WPSOffice_Level2) [12](#_Toc26852_WPSOffice_Level2)

[6、 watchFile()、unwatchFile()](#_Toc29947_WPSOffice_Level2) [12](#_Toc29947_WPSOffice_Level2)

[7、 createReadStream()](#_Toc11941_WPSOffice_Level2) [12](#_Toc11941_WPSOffice_Level2)

[8、 createWriteStream()](#_Toc23562_WPSOffice_Level2) [12](#_Toc23562_WPSOffice_Level2)

[11、Http模块](#_Toc485_WPSOffice_Level1) [12](#_Toc485_WPSOffice_Level1)

[1、http.STATUS\_CODES](#_Toc10276_WPSOffice_Level2) [12](#_Toc10276_WPSOffice_Level2)

[2、基本用法](#_Toc23721_WPSOffice_Level2) [13](#_Toc23721_WPSOffice_Level2)

[3、request对象](#_Toc29583_WPSOffice_Level2) [13](#_Toc29583_WPSOffice_Level2)

****

**1、Nodejs异常处理**

是单线程，即当出现异常报错的时候必须处理异常。一般来说有三种方法：

1. 使用throw语句抛出一个错误对象，即抛出异常。
2. 将错误对象传递给回调函数，由回调函数负责发出错误。
3. 通过EventEmitter接口，发出一个error事件。
4. 使用捕获异常方式就是 try…catch结构，但是此方法无法捕获异步运行的代码抛出的异常。
5. 使用 回调函数，将错误的对象作为第一个参数，传入 回调函数中，这样可以避免捕获的代码与错误的代码不在同一个时间段的问题。
6. 使用 EventEmitter接口的error事件。
7. UncaughtException事件，当一个异常未被捕获，就会触发此事件，即可以对这个事件注册回调函数，从而捕获异常。但是此方法发生异常时上下文已经丢失所以无法获取异常的详细信息。
8. Iojs有一个unhandleRejection事件，用来监听没有捕获的Promise对象的rejected状态。

**2、assert模块**

Assert模块是node的内置模块，主要用于断言。如果表达式不符合预期，就抛出一个错误。其有11个方法。

1. assert()：assert方法接受两个参数，当第一个参数对应的布尔值为true时，不会有任何提示，返回undefined。当第二个参数对应的布尔值为false时，会抛出一个错误，该错误提示信息就是第二个参数的字符串。
2. assert.ok()：与assert()用法完全相同。
3. assert.equal()：equal方法接受三个参数，第一个参数是实际值，第二个是预期值，第三个是错误提示信息。

assert(1==2,”错了”) 和 assert.ok(1==2,”错了”) 和 assert.equal(1,2,”错了”)

上面三个写法是一样的效果。

1. assert.notEqual()：notEqual方法与equal方法类似，但是只有在实际值等于预期值时，才会抛出错误。
2. assert.deepEqual()：用来比较两个对象，只要他们的属性一一对应，且值都相等，就认为两个对象相等，否则抛出异常。
3. assert.notDeepEqual()：与上面的相反，用来断言两个对象是否不相等。
4. assert.strictEqual()：此方法使用严格的相等运算符（===）,比较两个表达式。
5. assert.notStrictEqual()：与上面方法相反，严格不相等运算符（！==）。
6. assert.throws()：此方法预期某个代码块抛出异常，且抛出错误符合指定的条件。
7. assert.ifError()：此方法断言某个表达式是否false，如果该表达式对应的布尔值等于true，就抛出一个错误。它对于验证回调函数的第一个参数十分有用，如果改参数为true时，就表示有错误。
8. assert.fail()：fail方法用于抛出一个错误。

**3、Buffer**

Buffer对象是node处理二进制数据的一个接口。它是node原生提供的全局对象，可以直接使用，不需要require。

JavaScript比较擅长处理字符串，但是对于处理二进制数据（比如TCP数据流）就不太擅长。Buffer就是为了解决此问题而设计的。它是一个构造函数。

生成实例： var bytes = new Buffer(256);

Buffer对象与字符串的相互转换，需要制定编码格式。目前支持一下几种：

（1）、ascii

（2）、utf-8

（3）、base64

（4）、utf16le：UTF-16的小端编码，支持大于U+10000的四字节字符

（5）、hex：将每个字节转为两个十六进制字符

**Buffer的类方法**

（1）、Buffer.isEncoding()：返回一个布尔值，表示Buffer实例是否为指定编码。

（2）、Buffer.isBuffer()：返回一个布尔值，接受一个对象作为参数。

（3）、Buffer.byteLength()：返回字符串实际占据的字节长度，默认为UTF-8。

（4）、Buffer.concat()：将一组Buffer对象合并为一个Buffer对象。此方法还有第二个参数，指定合并后Buffer对象的总长度。

**实例属性：**

（1）、write()

Write方法可以向指定的Buffer对象写入数据，它的第一个参数是所写的内容，第二个参数（可省略）是所写入的起始位置（默认0开始），第三个参数是编码方式（可省略）。

（2）、slice()

Slice方法返回一个按照指定位置、从原对象切割出来的Buffer实例。它的两个参数分别是切割的起始位置和终止位置。

（3）、toString()

此方法将Buffer实例按照指定的编码（默认utf-8）转为字符串，可以返回指定位置内存的内容，它的第二个参数和第三个参数表示起始位置和终止位置，都是从0开始计算。

1. toJSON()

此方法将Buffer实例转换为JSON对象，如果JSON.stringify方法调用Buffer实例，默认会先调用toJSON方法。

**4、Child Process 模块**

child\_process模块用于新建子进程。子进程的运行借故偶存储在系统缓存中（最大200KB），等到子进程运行结束以后，主进程再用回调函数读取子进程的运行结果。

1、exec()

Exec方法用于执行bash命令，它的参数是一个命令字符串。

var exec = require(‘child\_process’).exec;

var ls = exec(‘ls -l’,callback);

exec方法最多接受两个参数，第一个是shell命令，第二个数回调函数，该函数可以接受三个参数，分别是：发生的错误、标准输出结果、标准错误的结果。（标准输出都是流对象）

2、execFile()

execFile方法直接执行特定的程序，参数作为数组传入，不会被bash解释，具有较高的安全性。

1. spawn()

spawn方法创建一个子进程来执行特定命令，用法与execFile方法类似，但是没有回调函数，只能通过监听事件来获取运行结果。属于异步执行。

1. fork()

fork方法直接创建一个子进程，与spawn方法不同的是，fork会在父进程与子进程之间建立一个通道，用于进程之间的通信。

1. send()

使用 child\_process.fork() 生成新进程之后，就可以用 child.send(message, [sendHandle]) 向新进程发送消息。新进程中通过监听message事件，来获取消息。

**5、文件查找策略**

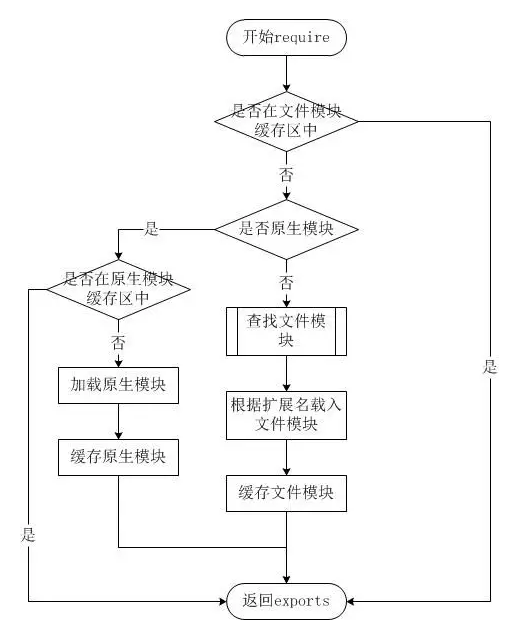
1、require()查找优先级的顺序：

文件模块缓存>原生模块缓存>原生模块>文件模块

当确定是文件模块的时候 nodejs会解析require方法传进来的参数，并从文件系统中加载实际的文件

当Node遇到require(X)时，按照下面的顺序处理：

1. 、如果 X 是内置模块（比如 require('http'）);
2. 返回该模块。
3. 不再继续执行
4. 、如果X以 “./”或者“/”或者“../”开头
5. 根据X所在的父模块，确定X的绝对路径。
6. 将X当成文件，一次查按照以下的顺序查找，只要存在即返回停止：
7. X ,-->2. X.js ,--> 3. X.json -->4. X.node
8. 将X当成目录，一次查找下面的文件，只要存在即返回停止：
9. X/package.json
10. X/index.js
11. X/index.json
12. X/index.node
13. 如果X不带路径
14. 根据X所在的父模块，确定X可能安装的目录。
15. 依次在每个目录中，将X当成文件名或目录名加载。
16. 抛出not found“”

整个流程如下图所示：

**6、cluster模块**

Node.js默认单进程运行，对于32位系统最高可以使用512MB内存，对于64位最高可以使用1GB内存。对于多核CPU的计算机来说，这样做效率很低，因为只有一个核在运行，其他核都在闲置。cluster模块就是为了解决这个问题而提出的。

cluster模块允许设立一个主进程和若干个worker进程，由主进程监控和协调worker进程的运行。worker之间采用进程间通信交换消息，cluster模块内置一个负载均衡器，采用Round-robin算法协调各个worker进程之间的负载。运行时，所有新建立的链接都由主进程完成，然后主进程再把TCP连接分配给指定的worker进程。

1. **worker对象**

Worker对象是cluster.fork()的返回值，代表一个worker进程。它的属性和方法如下：

1. 、worker.id

Worker.id返回当前worker的独一无二的进程编号。这个编号也是cluster.worker中指向当前进程的索引值。

1. 、worker.process

所有的worker进程都是用child\_process.fork()生成的，child\_process.fork()返回的对象，就被保存在worker.process之中。通过这个属性，可以获取worker坐在的进程对象。

1. 、worker.send()

该方法用于在主进程中向子进程发送信息。在worker进程中，要想主进程发送消息，使用process.send(message)，要监听主进程发送的消息，使用process.on()

1. **Cluster.workers对象**

该对象只有主进程才有，包含了所有worker进程。每个成员的键值就是一个worker进程对象，键名就是该worker进程的worker.id属性。（cluster.workers[id]）

1. **Cluster模块的属性与方法**
2. 、isMaster，isWorker

isMaster属性返回一个布尔值，表示当前进程是否为主进程。这个属性由process.env.NODE\_UNIQUE\_ID决定，如果process.env.NODE\_UNIQUE\_ID为未定义，就表示该进程是主进程。

isWorker属性返回一个布尔值，表示当前进程是否为work进程。它与isMaster属 性的值正好相反。

1. 、fork()

Fork()方法用于新建一个worker进程，上下文都是复制主进程。只有主进程才能调用这个方法。该方法返回一个worker对象。

1. 、kill()

kill方法用于终止worker进程。它可以接受一个参数，表示系统信号。

如果当前是主进程，就会终止与worker.process的联络，然后将系统信号法发 向worker进程。如果当前是worker进程，就会终止与主进程的通信，然后退 出，返回0。在以前的版本中，该方法也叫做 worker.destroy() 。

1. 、listening事件

worker进程调用listening方法以后，“listening”事件就传向该进程的服务器， 然后传向主进程。

该事件的回调函数接受两个参数，一个是当前worker对象， 另一个是地 址对象，包含网址、端口、地址类型（IPv4、IPv6、Unix socket、 UDP）等 信息。这对于那些服务多个网址的Node应用程序非常有用。

1. **不中断地重启Node服务**

重启服务需要关闭后再启动，利用cluster模块，可以做到先启动一个worker进程，再把原有的所有work进程关闭。这样就能实现不中断地重启Node服务。

首先，主进程向worker进程发出重启信号。

**workers[wid].send({type: 'shutdown', from: 'master'});**

worker进程监听message事件，一旦发现内容是shutdown，就退出。

**process.on('message', function(message) {**

**if(message.type === 'shutdown') {**

**process.exit(0);**

**}});**

1. **PM模块**

PM2模块是cluster模块的一个包装层。它的作用是尽量将cluster模块抽象掉，让用户像使用单进程一样，部署多进程Node应用。

**7、Node应用程序开发**

（1）、启动

在shell启动node脚本的时候通常使用：node example.js 但是，这个shell随着退出shell就会自动结束。为了node应用程序可以在后台长期运行，可以使用一下方法实现：

forever，

forever是一个node应用程序，用于一个子进程意外退出时自动重启。

forever start example.js、 forever stop example.js

**8、events模块**

回调函数模式让 Node 可以处理异步操作。但是，为了适应回调函数，一步操作只能有两个状态：开始和结束。对于那些多状态的异步操作（状态1，状态2，.....），回调函数就会无法处理，你不得不将异步操作拆开，分成多个阶段。每个阶段结束时，调用下一个回调函数。

为了解决这个问题，Node 提供 Event Emitter 接口。通过事件，解决多状态异步操作的响应问题。

1. 、概述

Event emitter 是一个接口，可以在任何对象上部署。这个接口又events模块提供。

var EventEmitter = require(‘events’).EventEmitter;

var emitter = new EventEmitter();

Events 模块的eventemitter 是一个构造函数，可以用来生成时间发生器的实例emitter。然后，事件发生器的实例方法on用来监听事件，实例方法emit用来发出事件。

emitter.on('someEvent', function () {

console.log('event has occured');});

function f() {

console.log('start');

emitter.emit('someEvent');

console.log('end');}

f();

运行结果： start

Event has occured

end

如上面的中，EventEmitter对象实例emitter就是消息中心。通过on方法为someEvent事件指定回调函数，通过emit方法触发someEvent事件。上面代码还表明，EventEmitter对象的事件触发和监听是同步的，即只有事件的回调函数执行以后，函数f才会继续执行。

1. 、Event Emitter 接口的部署

Event Emitter 接口可以部署在任意对象上，使得这些对象也能订阅和发布消息。

var EventEmitter = require('events').EventEmitter;

Dog.prototype.\_\_proto\_\_ = EventEmitter.prototype;

// 另一种写法

// Dog.prototype = Object.create(EventEmitter.prototype);

var simon = new Dog('simon');

simon.on('bark', function () {

console.log(this.name + ' barked');});

setInterval(function () {

simon.emit('bark');}, 500);

上面代码新建了一个构造函数Dog，然后让其继承EventEmitter，因此Dog就拥有了EventEmitter的接口。最后，为Dog的实例指定bark事件的监听函数，再使用EventEmitter的emit方法，触发bark事件。

Node 内置模块util的inherits方法，提供了另一种继承 Event Emitter 接口的方法。

util.inherits(Radio, EventEmitter);这样也可以实例继承EventEmitter接口。

1. 、Event Emitter的实例方法

方法如下：

Emitter.emit() emit方法用来触发事件。第一个参数是事件名，其他参数都会传入 回调函数中。

Emitter.on(name,fun) 对事件name指定监听事件fun

Emitter.addListener(name,fun) addListener是on方法的别名

Emitter.once(name,fun) 与on方法类似，但是监听函数fun是一次性的，使用后自 动移除。

Emitter.listeners(name) 返回一个数组，成员是事件name所有监听函数

Emitter.removeListener(name,fun) 移除事件name的监听函数fun，所以不能用于移 除匿名函数。

Emitter.removeAllListeners(name) 移除事件name的所有监听函数

Emitter.setMaxListeners() node默认允许同一个事件最多可以指定10个回调函 数。但是可以使用此方法来改变最大值。

1. 、错误捕获

1、事件处理过程中抛出的错误，可以使用try...catch捕获。

监听函数抛出的错误被try...catch代码一旦被捕获，该事件后面的监听函数都不会再执行了。

1. 也可以让进程监听uncaughtException事件来处理错误
2. 、事件类型

Events模块默认支持两个事件：

1）、newListener事件：添加新的回调函数时触发。

2）、removeListener事件：移除回调时触发。

9、Express框架

Express是目前最流行的基于Node.js的Web开发框架，可以快速地搭建一个完整功能的网站。

1. **运行原理**

Express框架建立在node.js内置的http模块上。Express框架的核心是对http模块的再包装。Express框架等于在http模块之上，加了一个中间层。

1. **中间件**

简单说，中间件（middleware）就是处理HTTP请求的函数。它最大的特点就是，一个中间件处理完，再传递给下一个中间件。App实例在运行过程中，会调用一系列的中间件。

每个中间件可以从App实例，接收三个参数，依次为request对象（代表HTTP请求）、response对象（代表HTTP回应），next回调函数（代表下一个中间件）。每个中间件都可以对HTTP请求（request对象）进行加工，并且决定是否调用next方法，将request对象再传给下一个中间件。

1. **use方法**

use是express注册中间件的方法，它返回一个函数。如果有多个中间件的话可以使用next()方法继续往下执行。如果不加则表示执行完当前的中间就就停止。

1. **Express的方法**

针对不同的请求，Express提供了use方法的一些别名。

all()方法

表示所有请求都必须通过该中间件，第一个参数可以使“\*”。

get()

get方法则是只有GET动词的HTTP请求通过该中间件，它的第一个参数是请求的 路径。由于get方法的回调函数没有调用next方法，所以只要有一个中间件被调 用了，后面的中间件就不会再被调用了。另外还有post()、put()、delete()方法。

set()

set方法用于指定变量的值。

**5、response对象**

（1）response.redirect方法

response.redirect方法允许网址的重定向。

response.redirect("/hello/anime");

response.redirect(301, "http://www.example.com");

（2）response.sendFile方法

response.sendFile方法用于发送文件。

response.sendFile("/path/to/anime.mp4");

（3）response.render方法

response.render方法用于渲染网页模板。

app.get("/", function(request, response) {

response.render("index", { message: "Hello World" });});

**6、动态模板引擎**

Express支持多种模板引擎，handlebars、ejs、jade等等。

1、设置静态文件的目录地址，指定一个文件为静态文件的目录：

App.use(express.static(‘public’))

2、加载页面模板：使用render来加载渲染页面

Response.render(‘index’,{})

10、fs模块

fs是filesystem的缩写，该模块提供本地文件的读写能力，基本上是POSIX文件操作命令的简单包装。但是，这个模块几乎对所有操作提供异步和同步两种操作方式，供开发者选择。

1. readFile()、readFileSync()

1)、readFile方法用于异步读取数据。

fs.readFile(‘./image.png’,function(err,buffer){

If(err) throw err

process(buffer)

})

readFile 方法的第一个参数是文件的路径，可以是绝对路径，也可以是相对路径。注意，如果是相对路径，是相对于当前进程所在的路径（process.cwd()），而不是相对于当前脚本所在路径。

readFile方法的第二个参数是读取完成后的回调函数。该函数的第一个参数是发生错误时的错误对象，第二个参数是代表文件内容的Buffer实例。

2)、readFileSync方法用于同步读取文件，返回一个字符串。

var text = fs.readFileSync(fileName, 'utf8');

// 将文件按行拆成数组

text.split(/\r?\n/).forEach(function (line) {

});

readFileSync方法的第一个参数是文件路径，第二个参数可以是一个表示配置的对象，也可以是一个表示文本文件编码的字符串。默认的配置对象是{ encoding: null, flag: 'r' }，即文件编码默认为null，读取模式默认为r（只读）。如果第二个参数不指定编码（encoding），readFileSync方法返回一个Buffer实例，否则返回的是一个字符串。

1. writeFile()，writeFileSync()

1)、writeFile方法用于异步写入文件。

fs.writeFile('message.txt', 'Hello Node.js', (err) => {

if (err) throw err;

console.log('It\'s saved!');

});

上面代码中，writeFile方法的第一个参数是写入的文件名，第二个参数是写入的字符串，第三个参数是回调函数。

回调函数前面，还可以再加一个参数，表示写入字符串的编码（默认是utf8）。

fs.writeFile('message.txt', 'Hello Node.js', 'utf8', callback);

2)、writeFileSync方法用于同步写入文件。

fs.writeFileSync(fileName, str, 'utf8');

它的第一个参数是文件路径，第二个参数是写入文件的字符串，第三个参数是文件编码，默认为utf8。

3、exits(path,callback)

exists方法用来判断给定路径是否存在，然后不管结果如何，都会调用回调函数。

4、mkdir()，writeFile()，readFile()

1)、mkdir方法用于新建目录。

fs.mkdir('./helloDir',0777, function (err) {

if (err) throw err;

});

mkdir接受三个参数，第一个是目录名，第二个是权限值，第三个是回调函数。

1. 、writeFile方法用于写入文件。

fs.writeFile(‘./helloDir/message.text’,’hello word’,function(err){

If(err) throw err;

})

3)、readFile方法用于读取文件内容。

fs.readFile(‘./helloDir/message.text’,’utf-8’,function(err,data){

If(err) throw err;

Console.log(data)

})

上面代码使用readFile方法读取文件。readFile方法的第一个参数是文件名，第二个参数是文件编码，第三个参数是回调函数。可用的文件编码包括“ascii”、“utf8”和“base64”。如果没有指定文件编码，返回的是原始的缓存二进制数据，这时需要调用buffer对象的toString方法，将其转为字符串。

readFile方法是异步操作，所以必须小心，不要同时发起多个readFile请求。

1. stat()

stat方法的参数是一个文件或目录，它产生一个对象，该对象包含了该文件或目录的具体信息。我们往往通过该方法，判断正在处理的到底是一个文件，还是一个目录。

1. watchFile()、unwatchFile()

watchfile方法监听一个文件，如果该文件发生变化，就会自动触发回调函数。

unwatchfile方法用于解除对文件的监听。

1. createReadStream()

createReadStream方法往往用于打开大型的文本文件，创建一个读取操作的数据流。所谓大型文本文件，指的是文本文件的体积很大，读取操作的缓存装不下，只能分成几次发送，每次发送会触发一个data事件，发送结束会触发end事件。

1. createWriteStream()

createWriteStream方法创建一个写入数据流对象，该对象的write方法用于写入数据，end方法用于结束写入操作。

createWriteStream方法和createReadStream方法配合，可以实现拷贝大型文件。

11、Http模块

1、http.STATUS\_CODES

http.STATUS\_CODES是一个对象，属性名都是状态码，属性值则是该状态码的简短解释。

require('http').STATUS\_CODES['301']

2、基本用法

http模块主要用于搭建HTTP服务。使用Node搭建HTTP服务器非常简单。

var http = require('http');

http.createServer(function (request, response){

response.writeHead(200, {'Content-Type': 'text/plain'});

response.write("Hello World");

response.end();}).listen(8080, '127.0.0.1');

console.log('Server running on port 8080.');

// 另一种写法

function onRequest(request, response) {

response.writeHead(200, {"Content-Type": "text/plain"});

response.write("Hello World");

response.end();}

http.createServer(onRequest).listen(process.env.PORT);

3、request对象

createServer方法的回调函数的第一个参数是一个request对象，拥有以下属性。

url：发出请求的网址。

method：HTTP请求的方法。

headers：HTTP请求的所有HTTP头信息。