1- 包

- 由于 Node是一套轻内核的平台，虽然提供了一系列的内置模块，
  但是不足以满足开发者的需求，于是乎出现了包（package）的概念
- 与核心模块类似，就是将一些预先设计好的功能
  或者说 API封装到一个文件夹，提供给开发者使用；

1-1包的加载机制：

- require("http")
- 先在系统核心（优先级最高）的模块找
- 再到当前项目中的node_modules目录中找

如何管理：NPM

NPM可以管理包括它本身在内的包

1-2安装NPM

NPM 不需要单独安装。默认在安装 Node 的时候，会连带一起安装 NPM。
但是，Node 附带的 NPM 可能不是最新版本，最好用下面的命令，更新到最新版本。

```
$ npm install npm -g
```

默认安装到当前系统 Node 所在目录下。

由于之前使用 NVM 的方式安装的 Node 所以需要重新配置 NPM 的全局目录

1-3常用NPM指令

指令	说明
npm config	配置npm
npm init	初始化npm
npm search	查找模块
npm install	安装模块
npm uninstall	卸载模块

npm list	勘察安装模块
npm outdated	检查模块是否过时
npm update	更新模块
npm cache [clean|ls]	管理模块缓存

2- 文件系统操作

内置文件模块：

模块名称	功能
fs	–基础的文件操作 API
path	提供和路径相关的操作 API
*readline	用于读取大文本文件，一行一行
fs-extra（第三方）	https://www.npmjs.com/pacl

路径操作模块（path）

- 使用物理路径（绝对路径）
- path.join([p1][,p2][,p3]…) => 连接多个路径
- path.dirname(p) => 获取文件路径
- path.extname(p) => 获取文件扩展名
- path.basename(p, ext) => 获取文件名
- path.delimiter; => 获取路径分隔符
- path.format(obj) => 将路径对象转换成字符串
- path.parse(p) => 将一个路径解析成一个对象的形式
- path.relative([from ...], to) => 获取从 from 到 to 的相对路径
- path.isAbsolute(lrc); => 判断路径是否为绝对路径
- path.normalize(p) = > 常规化一个路径

同步或异步调用：

- fs模块对文件几乎所有操作都有同步和异步两种形式
- readFile() 和 readFilesync()

+ 区别

- 同步调用会阻塞代码执行， 异步不会
- 异步调用会将读取任务下达到任务队列，任务执行完成才会回调
  - 处理异常，同步必须使用 try catch 方式，异步使用回调函数的第一个参数

文件操作

创建	查询	更新
异步写入 fs.writeFile(file, data[, options], callback)	异步文件读取 fs.readFile(file[,options],callback(err,data))	异步追加 fs.appendFile(file,data[,option callback(err))
同步写入 fs.writeFileSync(file, data[, options])	同步文件读取 fs.readFileSync(file,[,option])	同步追加 fs.appendFileSync(file,data[,o ons])
流文件 fs.createWriteStream(path[,opaions])	文件流的方式读取 fs.createReadStream(path[, options])	异步重命名/移动文件 fs.rename(oldPath,newPath,ca ack)
	获取文件信息 fs.stat(path,callback(err,stats))	同步重命名/移动文件 fs.renameSync(oldPath,newPa
	同步获取文件信息 fs.statSync(path)	

目录操作

创建	查询	更新
异步创建目录 fs.mkdir(path[,mode],callback)	异步读取目录文件夹和文件 fs.readdir(path,callback)	异步重命名/移动目录 fs.rename(oldPath,newPath,ca ack)
同步创建目录 fs.mkdirSync(path[,mode])	同步读取目录文件夹和文件 fs.readdirSync(path)	同步重命名/移动目录 fs.renameSync(oldPath,newPa

打开文件：fs.open(path, flags[, mode], callback)

参数：

- path - 文件的路径。
- flags - 文件打开的行为。具体值详见下文。
- mode - 设置文件模式(权限)，文件创建默认权限为 0666(可读，可写)。
- callback - 回调函数，带有两个参数如：callback(err, fd)。

flags	描述
r	以读取模式打开文件。如果文件不存在抛出异常。
r+	以读写模式打开文件。如果文件不存在抛出异常。
rs	以同步的方式读取文件。
rs+	以同步的方式读取和写入文件。

w	以写入模式打开文件，如果文件不存在则创建。
wx	类似 'w'，但是如果文件路径存在，则文件写入失败。
w+	以读写模式打开文件，如果文件不存在则创建。
wx+	类似 'w+'， 但是如果文件路径存在，则文件读写失败。
a	以追加模式打开文件，如果文件不存在则创建。
ax	类似 'a'， 但是如果文件路径存在，则文件追加失败。
a+	以读取追加模式打开文件，如果文件不存在则创建。
ax+	类似 'a+'， 但是如果文件路径存在，则文件读取追加失败。

r：读取模式
w：写入模式
d：追加模式
?x：如果命令不符合，则事件失败
?+：文件不存在则创建

获取文件信息：fs.stat(path, callback)

- 监视文件变化：
- fs.watchFile(filename[, options], listener(curr,prev))
- options:{persistent,interval}
- fs.watch(filename[,options][,listener])

3- 缓冲区

什么是缓冲区：
- 缓冲区就是内存中操作数据的容器
- 只是数据容器而已
- 通过缓冲区可以很方便的操作二进制数据
- 而且在大文件操作时必须有缓冲区

3-1 为什么要有缓冲区

- JS 是比较擅长处理字符串，但是早期的应用场景主要用于处理 HTML 文档，不会有太大篇幅的数据处理，也不会接触到二进制的数据。
- 而在 Node 中操作数据、网络通信是没办法完全以字符串的方式操作的
- 所以在 Node 中引入了一个二进制的缓冲区的实现：Buffer

3-2 创建缓冲区

1. 创建长度为4个字节的缓冲区

```
var buffer = new Buffer(4);
```

2. 通过指定数组内容的方式创建

```
var buffer = new Buffer([00,01]);
```

3. 通过指定编码的方式创建

```
var buffer = new Buffer('hello', 'utf8');
```

3-3 Node默认支持编码

- Buffers 和 JavaScript 字符串对象之间转换时需要一个明确的编码方法。下面是字符串的不同编码。

编码	说明
ascii	7位的 ASCII 数据。这种编码方式非常快，它会移除最高位内容。
utf8	多字节编码 Unicode 字符。大部分网页和文档使用这类编码方式
utf16le	2个或4个字节, Little Endian (LE) 编码 Unicode 字符。编码范围 (U+10000
ucs2	utf16le'的子集
base64	Base64 字符编码
binary	仅使用每个字符的头8位将原始的二进制信息进行编码。 在需使用 Buffer 的情况下，应该尽量避免使用这个已经过时的编码方式。 这个编码方式将会在未来某个版本中弃用
hex	每个字节都采用 2 进制编码

node不支持GBK，GBK2312

3-4 使用场景

- 图片转成base64编码
- 文字编码

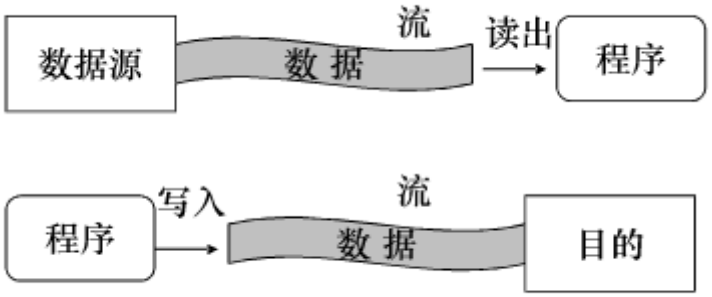
4 - 文件流

什么是流？

- 文件流、网络流
- 任何数据的最根本
- 表现形式都是二进制的

文件流：

- 文件流就是以面向对象的概念对文件数据进行的抽象
- 文件流定义了一些对文件数据的操作方式



Node.js 中有四种基本的流类型：

- Readable - 可读的流 (例如 fs.createReadStream()).
- Writable - 可写的流 (例如 fs.createWriteStream()).
- Duplex - 可读写的流 (例如 net.Socket).
- Transform - 在读写过程中可以修改和变换数据的 Duplex 流 (例如 zlib.createDeflate()).