黑马程序员
www.itheima.com

Node.js

讲师：李鹏周

行百里者半九十，前端工程师的逆袭

Node.js课程大纲

快速了解本课程知识内容

课程大纲

- Node.js简介
- 安装与配置
- 基础入门
- 模块与包
- 异步I/O与异步编程
- Buffer操作
- 文件I/O

课程大纲

- HTTP
- 使用Node构建Web应用
- Web开发框架（Express、koa）
- 数据库操作（MySQL、MongoDB）
- Socket.IO
- 多人博客案例
- 高级进阶

Node.js简介

行百里者半九十，前端工程师的逆袭

客户端JavaScript

- 知识回顾
- 什么是JavaScript？
- JavaScript是世界上最好的语言吗？
- 浏览器与JavaScript是什么关系？
- 浏览器中的JavaScript可以做什么？
- 浏览器中的JavaScript不可以做什么？
- JavaScript只可以运行在浏览器中吗？

服务器端JavaScript

- 使用JavaScript在浏览器中控制DOM元素作页面交互，这就是客户端JavaScript，因为它发生在浏览器或者客户端。

- 服务器端JavaScript发生在把页面发送给浏览器之前的服务器上，当然，使用的同样的语言！

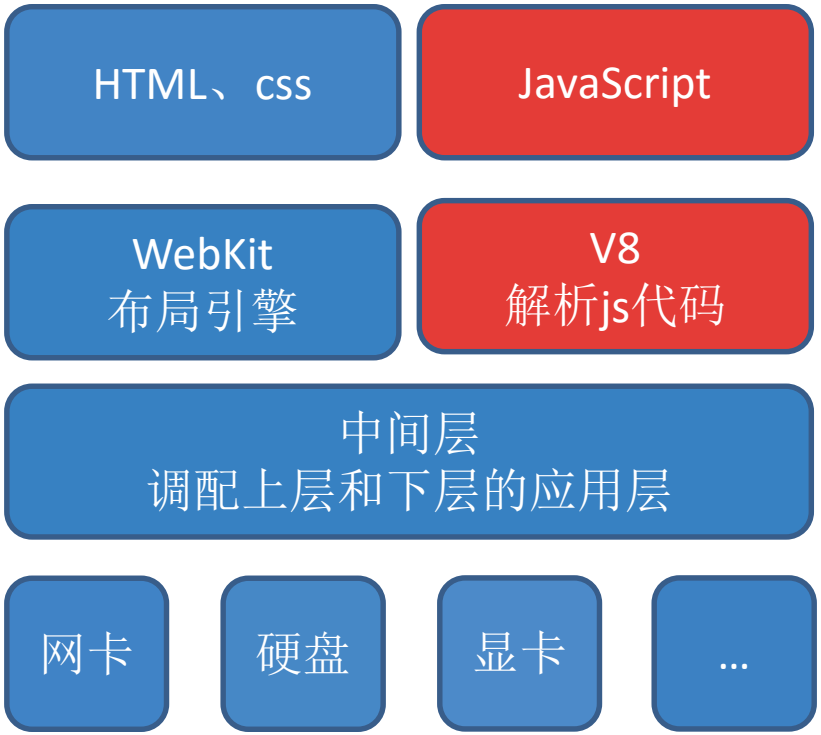
其它服务器端技术

- Java
- PHP
- .Net
- Ruby
- Python
- go

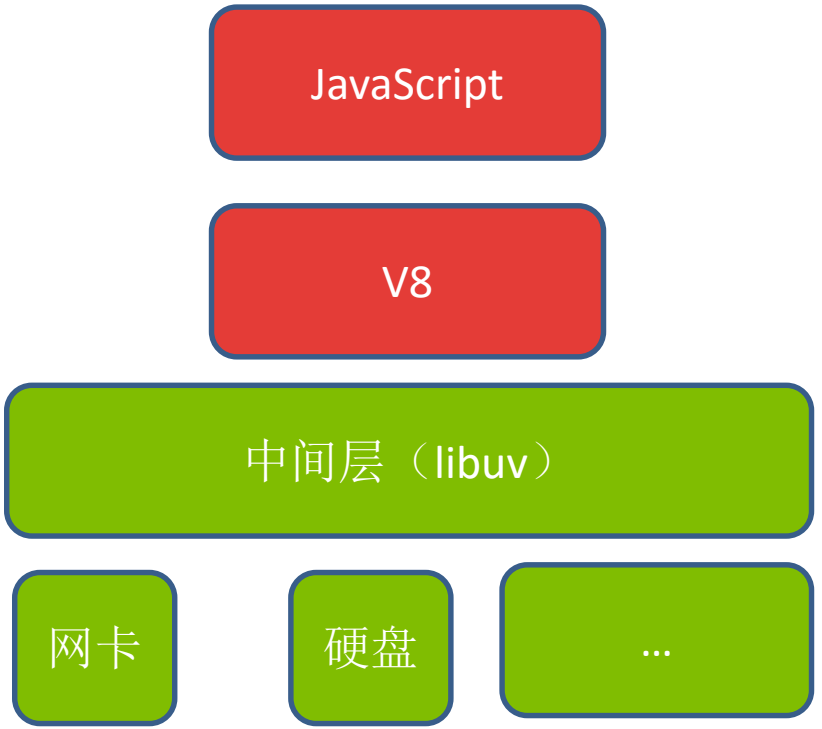
JavaScript的实现方式

浏览器	JavaScript实现方式
Firefox(火狐)	SpiderMonkey
IE	JScript
Safari	JavaScriptCore
Chrome	V8
Microsoft Edge	ChakraCore

Google Chrome



基于V8的Node横空出世



什么是Node.js？

- Node.js是一个在浏览器之外可以解析和执行JavaScript代码的运行时环境，或者说是一个运行时平台，理论意义上就是JavaScript语言在服务器端的运行环境
  - JavaScript 语言通过Node在服务器运行，在这个意义上，Node有点像 Java 虚拟机
  - Node提供大量工具库，使得 JavaScript 语言可以与操作系统互动（读写文件、网络IO、操作进程），在这个意义上，Node又是 JS 工具库
- Node.js的特性：**无阻塞IO模型、事件驱动**
- Node.js通常用来构建提供实时服务的应用程序。

关于Node

- 可以在服务器端使用JavaScript了
- 统一了开发环境和语言，JavaScript无处不在
- 高性能的JavaScript引擎 – Google V8
- 诞生于2009年，由Ryan Dasl 发布，并且是开源的
- Node.js非常轻量
- Node.js同时支持Windows、Linux、Mac OSX
- Node.js目前最新版本是4.4.3 | 5.11.0

Node诞生历程

- **Ryan Dahl**
- 2004年还在纽约**读数学系博士**
- 2006年退学，转战码农
- …接项目，去工作，旅行
- 2009年5月，正式对外宣布了Node.js的最初版本
- 专注于实现高性能Web服务器优化的专家，几经探索，几经挫折，遇到V8而诞生的项目

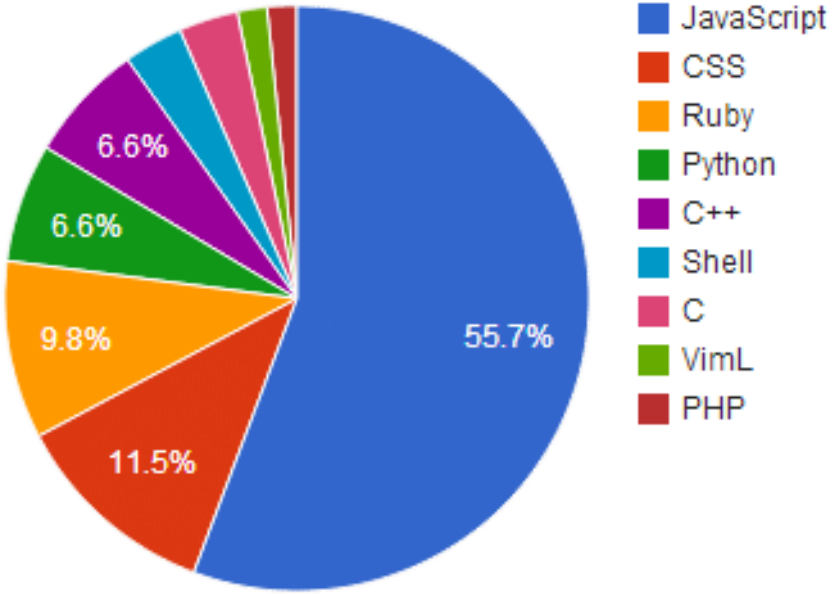
为什么要学习Node.js？

- 全栈开发工程师
  - 技能全面、学习能力强、沟通成本低、学习成本高
  - 掌握多种技能，独立完成产品
- 前端开发
  - html、css、JavaScript、jQuery、Angular、前端优化、
  - 自动化框架等
- 后端开发
  - Node.js构建后台服务
- 移动端开发
  - HTML5、ionic、React native、微信等。。。

为什么要学习Node.js？

- JavaScript已经是世界上最流行的开发语言
- 学习Node.js完全不需要重新学习一门新的语言
- 其它语言能做的，Node都可以做

  在某些场景下甚至更好



构建Node程序的各种解决方案（备注）

Node.js的可以做什么？

- 多人游戏、实时系统、联网软件和具有上千个并发用户的应用程序
- 实时多人游戏后台服务器
- 基于Web的聊天客户端
- 单页面浏览器应用程序
- 基于JSON的API
- 不适合CPU密集型应用

Node.js社区

- 09年诞生，作为一个第三方流行项目托管在github上
- node在github上目前有21386个star
- https://github.com/nodejs/node
- 从0.10.x开始每个月已经有超过两千万的下载了
- npm是目前全球最大的第三方包生态系统
- 目前有258032【2016-3-24】个包在npm上
- https://www.npmjs.com/

总结

- JavaScript只能在浏览器上运行吗？
  - JavaScript不仅仅能运行在浏览器中
- Node.js基于哪个JavaScript引擎？
  - Chrome 的 V8引擎
- 谁创建了Node.js？
  - **Ryan Dahl**
- Node.js的特性是什么？
  - 事件驱动、非阻塞IO模型
- Node.js是JavaScript吗？
  - Node.js不是JavaScript
  - Node.js是一个可以解析和执行JavaScript代码的运行时环境

安装与配置Node.js环境

介绍简单安装和使用nvm来管理多个版本的Node

在Windows下搭建node开发环境

- 官方网站：https://nodejs.org/en/
- 根据你的操作系统下载对应的软件包
- 安装
  - next
  - next
  - next
  - next...

版本管理工具nvm

- 项目地址：

  https://github.com/creationix/nvm

- 直接输入nvm查看nvm的常用命令以及作用
- nvm的一些常用命令：
  - 安装指定版本node　　nvm install 版本号 [arch]
  - 卸载指定版本node　　nvm uninstall 版本号
  - 切换使用指定版本的node　nvm use 版本号 [arch]
  - 查看本地安装的所有版本　nvm list|ls

path环境变量

- 当要求系统运行一个程序而没有告诉它程序所在的完整路径时
  - 系统首先在当前目录下面寻找该程序
  - 如果找不到，则系统会跑到path中指定的路径去找，如果找到，直接运行
  - 如果最终path环境变量中也没有找到，则直接提示不是内部或外部命令，也不是可运行的程序
- path环境的添加的两种方式
  - 直接在path的变量值中以分好分隔加入程序所在的目录
  - 也可以在外部先定义一个变量，然后在path以%变量名%的方式添加变量

cmd

- cmd：command 命令行控制台，允许用户可以在终端命令台中与操作系统交互，其实就是输入与输出（输入一些命令，输出一些结果）
- 作用：输入一些命令，cmd.exe可以执行，
- 在cmd中操作文件目录
  - cd（change directory） 切换目录
  - mkdir（make **directory**） 别名 md 创建一个文件夹
  - rd（remove directory） 别名 rm 删除文件夹
  - del（delete） 删除指定文件
  - dir 别名 ls 列出当前目录中所有的内容
  - ren（rename） 改变文件名
  - cls|clear（clear screen） 清屏

快速体验

1. 在命令行中输出hello world
2. 开发一个Web应用程序，输出hello world

```
var http = require('http');
http.createServer(function(req,res){
  res.end('hello world');
}).listen(3000);
```

Node.js基础

了解Node中全局作用域及全局对象和函数

REPL（Read-eval-print-loop）

- 作用
  - 方便测试JavaScript代码的运行环境
- REPL基本操作
  - 变量、函数、对象
  - 直接运行函数
  - 使用下划线字符，表示上一个命令的返回结果
- REPL基本命令
  - .help .exit

全局对象global

- global表示Node所在的全局环境，类似于浏览器的window对象
- 使用REPL环境查看global对象
- 注意：在REPL中定义的变量默认就是全局
- 总结：
  - global就表示Node中的全局命名空间，任何全局变量、函数或对象都是global的一个属性
  - 在一个模块中定义的变量、函数或方法只在该模块中可用，但可以通过exports对象将其传递到模块外部

Global

- __dirname和__filename
- setInterval()和clearInterval()
- setTimeout()和clearTimeout()
- console
- exports和module
- process
- require()
- Class:Buffer

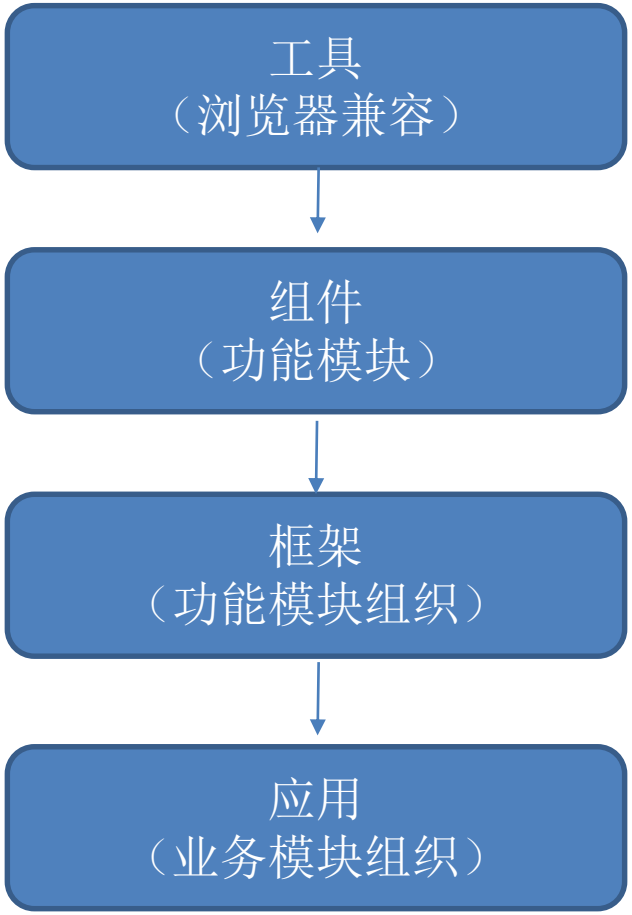
process

- process对象是Node的一个全局对象，提供当前Node进程的信息。可以在脚本的任意位置使用。
- stdout和stdin
- process.version
- process.uptime()
- process.platform
- process.nextTick(callback,[,arg][],…)
- process.kill(pid,[,signal])
- process.env

模块系统

Node.js中的模块化系统

JavaScript发展历史



模块化开发演变方案

- 模块化演变计算器案例
- 所谓的模块化开发其实也就是如何有效的组织你的JavaScript代码
  - 全局函数直接写到html中
  - 将js代码提取出来放到单独的js文件中
  - 封装对象的方式解决全局函数面临的问题
  - 给对象加入独立的作用域空间

CommonJS规范

- --希望JavaScript可以在任何地方运行，以达到像Java、PHP、Ruby、Python具备开发大型应用的能力
- 出发点：
  - 没有模块系统
  - 标准库较少
  - 缺乏包管理系统

CommonJS模块规范

- 一个单独的文件就是一个模块，每一个模块都是一个单独的作用域
- 1. 模块引用require
- 2. 模块定义
  - 一个文件就是一个模块
  - 将方法挂载到exports对象上作为属性即可定义导出的方式
- 3. 模块标识
  - 必须是符合小驼峰命名的字符串
  - 以.、..开头的相对路径
  - 绝对路径
  - 可以没有文件名后缀.js

CommonJS模块特点

- 所有代码都运行在模块作用域，不会污染全局作用域
- 模块可以多次加载，但是只会在第一次加载时运行一次，然后运行结果就缓存了，以后再加载，就直接读取缓存结果。
- 模块加载的顺序，按照其在代码中出现的顺序

Node.js模块介绍

- Node程序由许多模块组成，每个模块就是一个文件。Node模块采用了CommonJS规范。
- Node.js本身就是一个**高度模块化的一个平台**
- 根据CommonJS规范，每一个模块都是一个单独的作用域
- CommonJS规定，每个文件对外的接口是module.exports对象，该对象所有属性和方法，都可以被其它文件导入。

module对象

- Node内部提供一个Module构造函数，所有模块都是Module的实例
- 每个模块内部，都有一个module对象,代表当前模块。
  - module.id　　带有绝对路径的模块文件名
  - module.filename　　模块的文件名，带有绝对路径
  - module.loaded　　表示模块是否已经完成加载
  - module.parent　　返回一个对象，表示调用该模块的模块。
  - module.children　　返回一个数组，表示该模块要用到的其他模块。
  - module.exports　　模块对外输出的值

模块内的module.exports

- module.exports属性表示当前模块对外输出的接口，其它文件加载该模块，实际上就是读取module.exports属性
- 点儿导出单个函数、对象或者值的时候非常有用，本质上就是少了一个 •

模块内的exports

- 为了方便，Node为每个模块提供一个exports变量，指向module.exports。
- 相当于在每个模块头部，有这样一行命令：

```
var exports = module.exports;
```

- 结果就是：
  - 在对外输出模块接口时，可以向exports对象添加方法
  - 注意：不能直接给exports赋值，因为这样等于切断了exports和module.exports的联系

require()加载模块

- 在Node.js中，require命令用于加载模块文件

- 基本功能：
  - 读取并执行一个JavaScript文件
  - 然后返回该模块的exports对象
  - 如果没有发现指定模块,会报错

require模块加载规则

- 参数字符串以 “/” 开头
- 参数字符换以 “ . / ” 开头
- 参数字符串不以 “ . / “或” / ” ，表示加载核心模块，或者一个位于各级node_modules目录已安装的模块
- 参数字符串可以省略后缀名
  - .js、.json、.node
  - .js会当做JavaScript脚本文件解析
  - .json会以JSON格式解析
  - .node会以编译后的二进制文件解析

核心模块与文件模块

- 核心模块
  - require(‘核心模块名’)
- 文件模块
  - require(‘路径+模块名’)
  - 相对路径与绝对路径
  - 前缀“/”（类Unix操作系统与Windows的区别）
- 总结
  - 加载模块时将运行模块文件中的每一行代码
  - 相同模块多次引用不会引起模块内代码多次执行

核心模块

模块名称	功能
http	提供http服务器功能
url	解析url
fs	与文件系统交互
querystring	解析url查询字符串
util	提供一系列实用小工具
path	处理文件路径

核心模块的源码都在Node的lib子目录中。为了提高运行速度，它们安装的时候都会被编译成二进制文件

模块加载机制

- 如果require绝对路径的文件，查找时不会去遍历每一个node_modules目录，其速度最快。其余流程如下：
- 1. 从module path数组中取出第一个目录作为查找基准。
- 2. 直接从目录中查找该文件，如果存在，则结束查找。如果不存在，则进行下一条查找。
- 3. 尝试添加.js、.json、.node后缀后查找，如果存在文件，则结束查找。如果不存在，则进行下一条。
- 4. 尝试将require的参数作为一个包来进行查找，读取目录下的package.json文件，取得main参数指定的文件。
- 5. 尝试查找该文件，如果存在，则结束查找。如果不存在，则进行第3条查找。
- 6. 如果继续失败，则取出module path数组中的下一个目录作为基准查找，循环第1至5个步骤。
- 7. 如果继续失败，循环第1至6个步骤，直到module path中的最后一个值。
- 8. 如果仍然失败，则抛出异常。

练习

- 自己创建一个文件模块，实现一个加法计算器，可以被外部模块加载过后直接使用

- 自己创建一个包，并引入该包，注意包的结构以及package.json文件的使用

模块总结

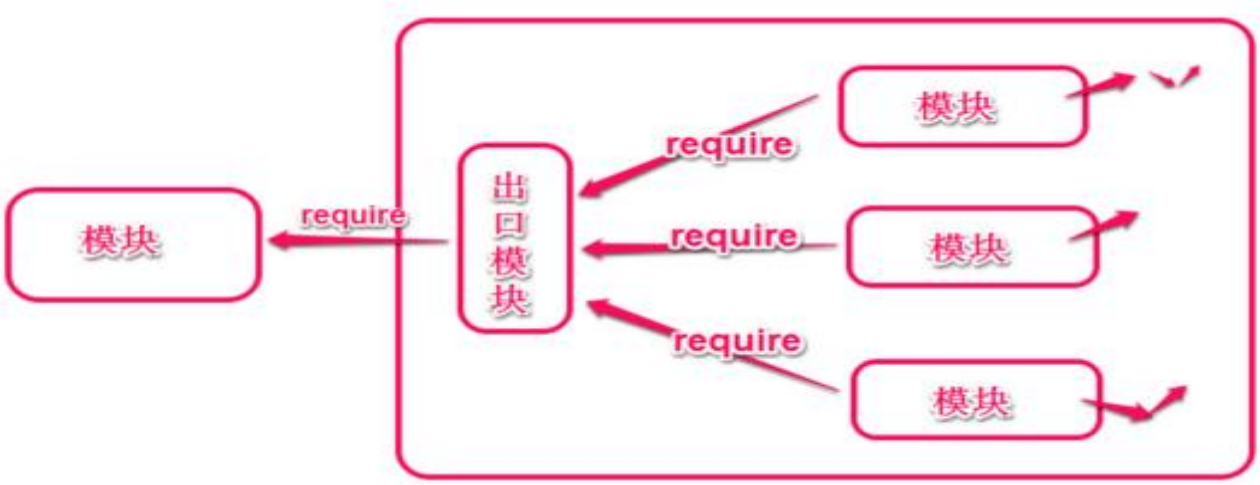
- 所有代码都运行在模块作用域，不会污染全局作用域
- 模块可以多次加载，但是只会在第一次加载的时候运行一次，然后运行结果就被缓存了，以后再加载，就直接读取缓存结果
- 模块的加载顺序，按照代码的出现的顺序是同步加载的
- require是同步加载模块的

包与NPM

熟练掌握并使用NPM包管理系统

包

- 什么是包？
- 包用来解决什么问题？
  - 将一堆的文件模块联系起来的一种机制
  - 在模块的基础上进一步组织JavaScript代码



规范的包目录结构

规范的包结构	作用
package.json	包描述文件，说明文件
Bin	存放可执行二进制文件的目录
Lib	存放JavaScript代码的目录
Doc	存放文档的目录
Test	存放单元测试用例的代码

规范带来的好处：

大家都不要乱来了，都遵守这个规范，

当看到一个规范的包目录的时候，会给你一种**踏实的感觉**

包描述文件package.json说明书

属性	作用
name	包的名称
description	包的简介
version	包的版本号
keywords	关键词数组，用于在npm中分类搜索
author	包的作者
main	配置包的入口，默认是模块根目录下的index.js
dependencies	包的依赖项，npm会通过该属性自动加载依赖包
scripts	指定了运行脚本命令的npmm命令行缩写，例如start

npm（Node.js package manage）

- 两种含义：
- 一种含义是Node.js的开放式模块登记和管理系统
- https://www.npmjs.com/
- 全球之最：最大的模块生态系统，里面所有的模块后者说是包，都是开源免费的，拿来即用
- 另一种含义是Node.js默认的模块管理器，是一个命令行下的软件，用来安装和管理node模块

NPM

- https://www.npmjs.com/
- 全球最大第三方模块生态系统
- 拿来主义的天堂
- 包规范只是理论，NPM是包规范的一个实现
- 作用：**安装和管理node模块**
- 安装npm
- 通过 npm help 查看命令列表
- npm -l 查看各个命令的简单用法

NPM基本使用

命令	作用
npm init 【-y】	初始化一个package.json文件
npm install 包名	安装一个包
npm install –save 包名	将安装的包添加到package.json的依赖中（dependencies）
npm install –g 包名	安装一个命令行工具
npm docs 包名	查看包的文档【非常有用】
npm root -g	查看全局包安装路径
npm config set prefix “路径”	修改全局包安装路径
npm list	查看当前目录下安装的所有包
npm list -g	查看全局包的安装路径下所有的包
npm uninstall 包名	卸载当前目录下某个包
npm uninstall –g 包名	卸载全局安装路径下的某个包
npm update 包名	更新当前目录下某个包

npm install-01

- 全局工具安装 npm install -g 包名
  - 全局安装一般用于安装命令行工具模块，http-server
- 本地项目依赖安装 npm install 包名
  - 将一个模块下载到当前项目的node_modules子目录
  - 只有在该项目目录中，才可以使用这些包
  - npm install 之前，会先检查node_modules目录中是否已存在该模块，如果存在，就不再重新安装了。
  - npm install 包名 -f或者–force 强制重新安装

npm install-02

- npm install --save 包名 与 dependencies 字段
- npm install 与 package.json中的dependencies
  - 自动找描述文件中的dependencies字段中的值，一个一个安装
- 安装指定版本：npm install 包名@版本号
  - npm 默认会安装最新稳定版

npm全局工具问题

- 假如在4.3.2版本的node中通过npm安装了一个http-server工具，切换到5.7.0之后发现http-server无法使用了。
- 解决方法：
  - 1. 修改npm的全局安装路径 npm config set prefix “”
  - 2. 将该路径添加到path环境变量中

CNPM

- 淘宝NPM镜像：http://npm.taobao.org/
- 与官方NPM的同步频率目前为10分钟一次
- 安装：`npm install -g cnpm --registry=https://registry.npm.taobao.org`
- 安装包：`cnpm install 包名`
- 其它命令基本一样，一般在安装包的时候使用它就可以了

nrm

- npm的问题：
  - 资源都在国外，有时候会被墙，导致无法下载或者很慢
- 作用：切换和管理镜像源
- 项目地址：

  https://www.npmjs.com/package/nrm
- 安装：`npm install -g nrm`

总结

- 包就是在模块的基础之上进一步组织JavaScript代码
- 模块的全局安装一般是安装工具性的东西，安装完成后可以使用该工具，例如bower、gulp、http-server
- 模块的本地局部安装一般是在项目开发中使用到的功能性模块，和具体的代码相关。例如request、
- 一个规范的包应该都包含一个package.json文件

Node.js的作用

I/O的意义
Node.js想要解决的问题
并发的意义
实现并发的不同方法

Node.js官方介绍

- Node.js是构建在Chrome V8 引擎之上的一个**平台**
- Node.js使用事件驱动的、非阻塞的I/O模型，这让其既轻量又高效。
- Node.js的包生态系统npm，是世界上最大的开源库生态系统

理解I/O（input/output）

- I/O【input/output】可以理解为一次输入和一次输出之间数据的移动
  - 使用键盘敲入内容（输入）并在屏幕上看到内容的显示（输出）
  - 移动鼠标（输入）并在屏幕上看到鼠标的移动（输出）

- I/O思想示例：在控制台中输入`echo ‘hello world’`
  - 从数据移动角度解释上面示例所发生的事情

```
PS C:\Users\iroc> echo 'hello world'
hello world
PS C:\Users\iroc>
```

处理基本输入

- 示例：京东用户注册
- 说明当用户点击注册的时候出错的可能

- 在该场景中，用户的输入是可以预测的，完全可以按照被预测的顺序编写程序

处理超过一个的输入

- 计算机程序可以接受超过一个的输入
- 例如：右图中的小霸王
- 上面示例中是一个用户和一个表单
- 现在是：
  - 两名玩家
  - 两个手柄，每个手柄8个按钮

- 该示例中：要想解决所有可能发生的场景就是一件巨大的任务，要精确预测用户玩游戏的方式及顺序就不容易了

处理海量用户的输入

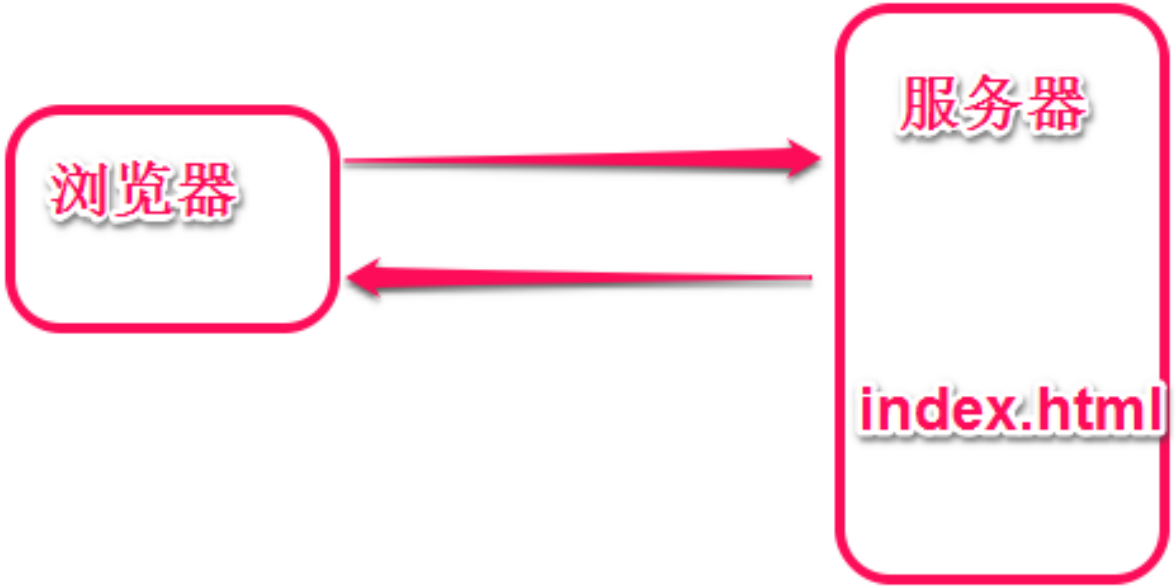
- 上百万的玩家
- 上百万个键盘
- 上百万个耳麦
- 玩家在3D虚拟世界中的各种操作
- 要想识别出每件可能发生的事情及顺序就成了不可能的任务
- 网络中的IO是及其复杂的、不可预测

早期的Web页面

- 世界上第一个网站：

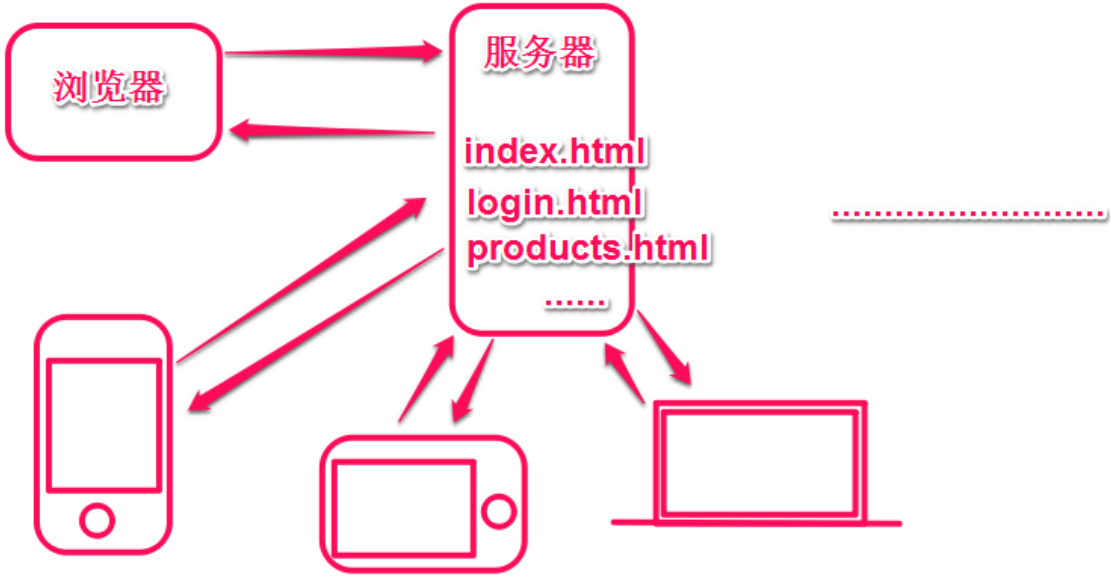
  http://info.cern.ch/hypertext/WWW/TheProject.html

- 功能简单，仅仅是知识的分享



现代的Web应用程序

- 现代Web应用程序的I/O是碎片化的、I/O操作更加频繁
  - 许多不同的设备发送与接收数据，移动设备、平板电脑、电视等
  - 巨大数量的客户同时连接并实时交互
- 如何高效的解决输入和输出就是Node.js解决的问题



联网的I/O不可预测

- 演示与时间有关的I/O不可预测性：
  - 使用http模块的get方法获取bat【baidu,qq,sina】的响应时间
  - 这里的输入是来自三个不同Web服务器的响应。Node.js将输出发送到终端
  - Web服务器响应时间会随着如下因素的某些因素的不同而变得极为不同【见备注2】
- 总结：基于网络的I/O不可预测

人类的不可预测

- 在开发Web页面交互的时候，要说出人类执行某个动作的顺序和时间是不可能的
- 示例1：给一个按钮注册点击事件
- 总结：我们并不是对一组用户可能进行的动作按线性排序列出而构建代码，而是围绕事件来构建代码
- 事件可以在任何时刻发生，也可能发生不止一次。我们将此描述为**事件驱动编程**。
- 用户什么时候输入不确定、输入了什么数据不确定

处理不可预测性

- 许多不同类型设备可以连接到Web应用程序
- 设备可以作为输入和输出
- 用户与服务器之间的数据实时传递
- 以上所有这些都指向了一个词：**并发**
- 并发：**输入会在同时发生并可能互相交互**
- Node.js将JavaScript解决不确定性所使用的事件驱动方式引入了进来。因为JavaScript是一种事件驱动的语言，旨在能够对外界的事件作出响应

小结

- 了解基本的I/O思想以及现代Web应用程序中输入和输出的数量如何巨大
- 在软件开发中要按时间和顺序预测人类行为是困难的
- JavaScript如何通过事件驱动的方式来响应的思想
- 并发的思想
- Node.js主要想解决的问题
- 思想总结：并发是软件开发中一直存在的问题，Node.js是对该问题的一个响应，尤其是在网络环境中

Node调试

任何一个平台的开发都离不开调试
调试不仅仅是工具
一个优秀的开发人员->调试能力很重要

调试

- 任何一个平台的开发都离不开调试
- 找到并去除缺陷的过程
  - 系统化注释掉或禁用代码块
  - 分析网络数据流，确定问题是客户端还是服务器
  - 使用之前能用的输入，并一点一定地修改输入，直到问题呈现
  - 用版本控制逐次回退，直到问题消失

Node调试-console.log()

- 最方便也最简单的：

  –`console.log()`

Node调试-node内置调试器

- 启动调试：node debug hello.js

- 常用命令
  - help 查看可用命令列表
  - n（下一步），s（步入），o（步出）

Node调试-node-inspector

- 一个第三方调试工具：node-inspector
- https://www.npmjs.com/package/node-inspector
- 安装：npm install -g node-inspector
- 1. 启动调试器：node-inspector，保持挂起不要关闭
- 2. 打开另一个命令台，以调试模式启动程序：
  - node --debug foo.js
  - node --debug-brk foo.js 调试器会在程序的第一行停住
- 3. 访问：http://localhost:8080/debug?port=5858

Node调试-node inspector使用

- 设置断点：单击行号
- 恢复脚本执行：F8
- 经过下一个函数调用：F10
- 进入下一个函数调用：F11
- 步出当前函数：Shift+F11
- 监视变量
- 使用控制台，探查变量，调用函数，建议不要过多使用这种方式动态修改运行中的程序，太容易迷糊
- 调试异步函数

Visual Studio Code调试

- 打开要调试的文件，按f5,编辑器会生成一个launch.json
- 修改launch.json相关内容，主要是name和program字段（要启动调试的文件）
- 点击编辑器左侧长得像蜘蛛的那个按钮
- 点击左上角DEBUG后面的按钮，启动调试
- 打断点，尽情调试（只要你会chrome调试，一模一样）

WebStorm调试

- 在要调试的脚本中打好断点之后右键选择Debug即可开启调试
- F8 Step over
- F7 Step into
- Shift + F7 Smart step into
- Shift + F8 Step out
- Alt + F9 Run to cursor
- Alt + F8 Evaluate expression
- F9 Resume
- Ctrl + F8 Toggle breakpoint
- Ctrl+Shift+F8 View breakpoints

异步I/O

了解Node中的异步异步I/O

异步操作

- Node采用Chrome V8引擎处理JavaScript脚本，V8最大的特点就是单线程运行，一次只能运行一个任务，代码从上到下按顺序执行
- Node大量采用异步操作，即任务不是马上执行,而是插在任务队列的尾部，等到前面的任务运行完后再执行
- 异步IO也叫非阻塞IO，例如读文件，传统的语言大部分都是读取完毕才能进行下一步操作。非阻塞就是Node的callback，不会影响下一步操作，等到文件读取完毕，回调函数自动被执行。而不是在等待。

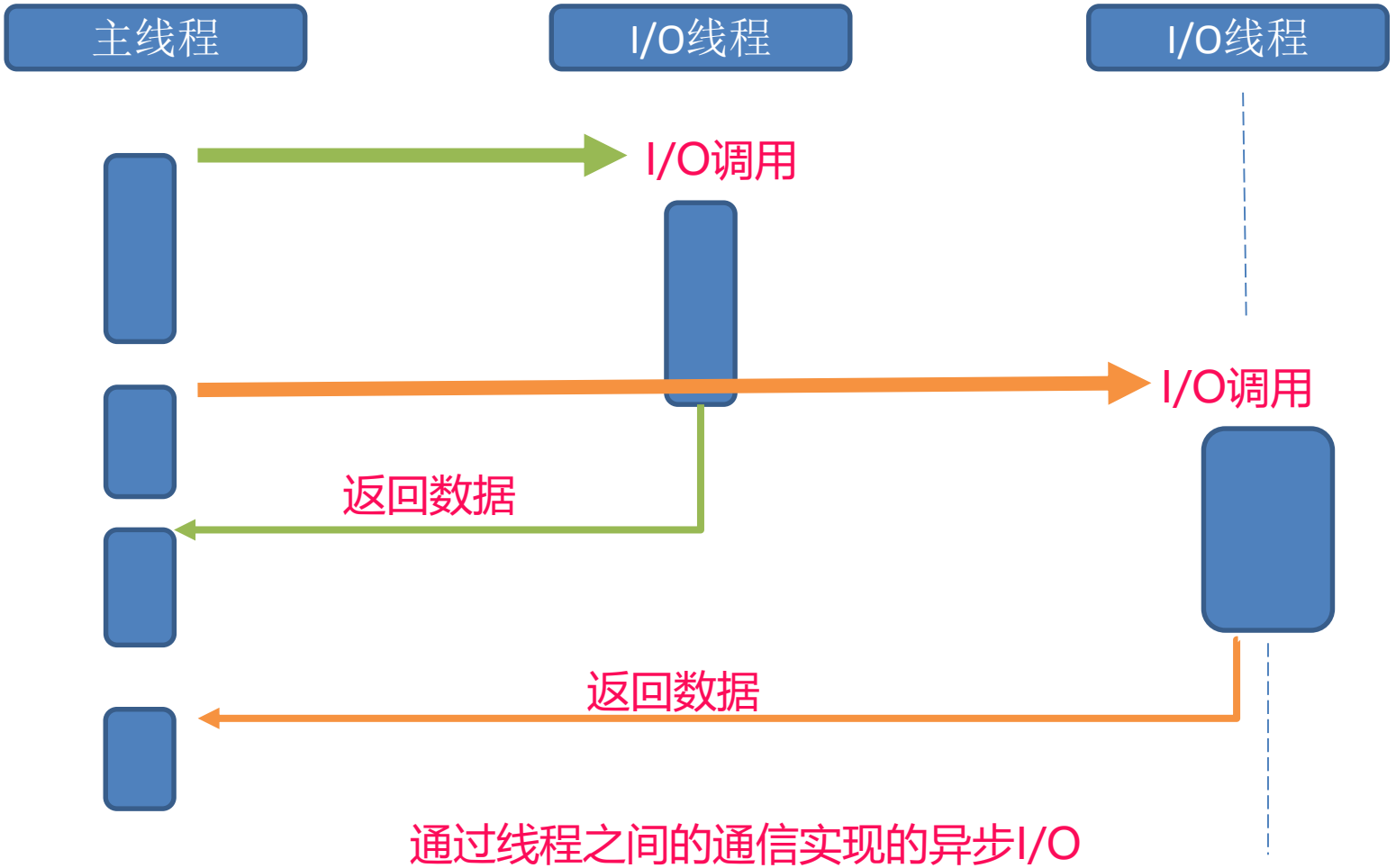
什么是进程

- 每一个正在运行的应用程序都称之为进程
- 每一个应用程序都至少有一个进程
- 进程是用来给应用程序提供一个执行的环境
- 进程是操作系统给应用程序分配资源的最小单位

什么是线程

- 用来执行应用程序中的代码
- 在一个进程内部，可以有很多线程
- 在一个线程内部,同时只能干一件事，代码从上到下依次执行
- 而且传统的开发方式大部分都是IO阻塞的
- 所以需要多线程来更好的利用硬件资源
- 给人带来一种错觉：线程越多越好

Node中的的异步I/O

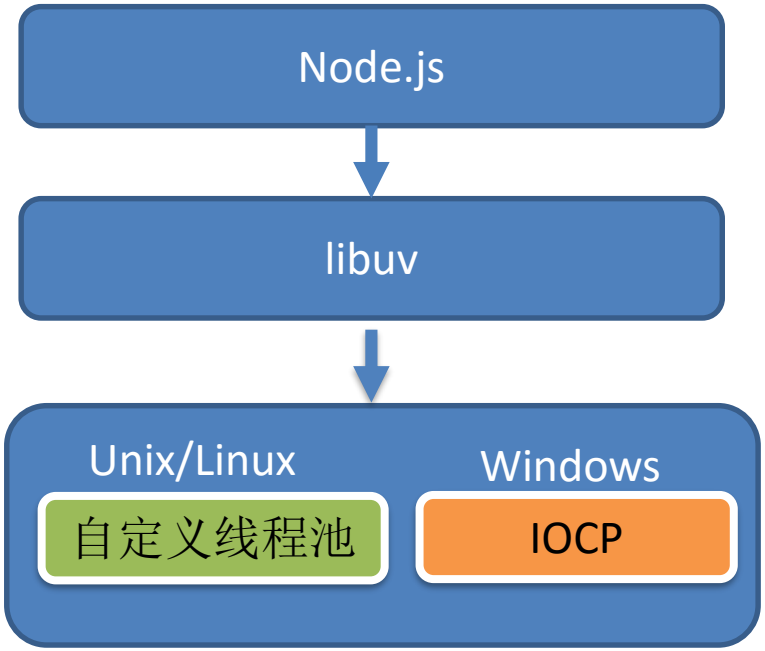


Node的异步I/O-线程池

- 注意：Node是单线程的，这里的单线程仅仅只是JavaScript执行在单线程中。在Node中，无论是类Unix还是Windows平台，内部完成I/O任务的有线程池，主要用来执行IO操作
- Node在Windows下使用的异步解决方案是Windows下的IOCP：调用异步方法，等待I/O完成之后的通知，执行回调，用户无需考虑轮询。但是它的内部其实仍然是线程池原理，不同之处在于这些线程池由系统内核接受管理
- Node在Unix/Linux平台下，0.9.3版本之前使用的是libeio配合libev实现的异步I/O，在0.9.3中，自行实现了线程池来完成异步I/O

Node的异步I/O-平台差异

- 由于Windows和Unix/Linux平台的差异，Node提供了libuv作为抽象封装层，保证上层的Node与下层的自定义线程池及IOCP之间各自独立



阻塞I/O

- 调用阻塞I/O时，应用程序需要等待I/O完成才返回结果
- **阻塞I/O特点**：调用之后一定要等到系统内核层面完成所有操作后，调用才结束，以读取磁盘上一段文件为例，系统内核在完成磁盘寻道、读取数据、复制数据到内存中之后，这个调用才结束
- 阻塞I/O造成CPU等待I/O，浪费等待时间，CPU的处理能力不能得到充分利用。

非阻塞I/O

- 为了提高性能，内核提供了非阻塞I/O
- 非阻塞I/O跟阻塞I/O的差别为**调用之后会立即返回**

- 阻塞I/O完成整个获取数据的过程，而非阻塞I/O则不带数据直接返回，要获取数据，还需要通过**文件描述符**再次读取
- 非阻塞I/O返回之后，CPU的时间片可以用来处理其它代码，此时性能提升是很明显的。

与IO无关的伪异步API

- setTimeout(callback,delay[,arg][,…])
- setInterval(callback,delay[,arg][],…)
- 上面两个有时间概念
- process.nextTick(callback[,arg][,…])
  - 一旦当前事件循环完成之后，下一次一开始就调用它
- setImmediate(callback[,arg][,…])
  - 同process.nextTick，但是优先级没有它高

异步编程带来的难点

- 异常处理：异步I/O包含两个阶段，提交请求和处理结果，这两个阶段中有事件循环的调度，两者彼此不关联。异步方法通常在第一个阶段提交请求后立即返回，因为异常并不一定发生在这个阶段，try/catch在此处不会发挥任何作用
- 函数嵌套过深 {{{{{{{{{{
- 多线程编程
  - 开发人员要面临跨线程通信编程
  - child_process
  - cluster

总结

- nodejs 是单线程的，这个单线程指的就是主线程，主线程不能异步，只能顺序执行；
- 但是主线程可以调用线程池来实现并发，线程池里的任务执行完成后发送一个事件到事件队列，事件循环会不断检测事件队列，发现有未处理的事件就分别调用它们的 callback 函数；
- callback 函数是顺序执行的，如果一个 callback 函数耗时很长，会阻塞事件循环，所以耗时很长的操作比如 IO 操作应该放在线程池里面执行；
- 主线程自始至终都是在事件循环中，主线程中的代码都是顺序执行，但是把耗时操作放在线程池中，然后写上 callback 函数，主线程的代码会继续向下执行，而事件循环会在适时的时候调用 callback 函数。所以在后面的代码可能比在前面的先执行完。

回调

回调是什么，它们在JavaScript中如何使用
Node.js中如何使用回调
同步和异步编程的区别
事件循环

什么是回调

- 程序角度：将一个函数作为参数传递给另一个函数，并且通常在第一个函数执行完成后被调用
- 回调示例：
  - 1．使用jQuery在网页中以动画的形式隐藏一个盒子【不使用回调】
  - 2．修改上面的示例【使用回调】

剖析回调

- 关键概念：函数可以作为参数传递到另一个函数中，然后被调用
- 剖析回调示例：【见备注】

- 理解回调的意义：因为这样的回调模式在Node.js中被到处使用

Node.js如何使用回调

- Node.js到处使用回调，尤其是在有I/O（输入/输出）的地方
- 示例一：使用核心模块fs的readFile方法读取文件【备注1】
- 示例二：使用第三方模块request访问一个url获取url的页面内容【备注示例2】
- 示例三：结合上面两个示例访问两个本地文件+请求两个url地址，看看哪个操作先返回【备注示例3】
- 多运行几次示例三，思考这些事件是如何异步发生的，回调是如何用于在操作完成后做其它事情的

异常处理

- Node是单线程运行环境，一旦抛出的异常没有被捕获，就会引起整个进程的崩溃。
- Node的异常处理对于保证系统的稳定运行非常重要。
- Node有三种方式，传播一个错误
  - throw抛出异常
  - 将错误对象传递给回调函数，由回调函数负责发出错误

try...catch捕获异常

- 使用try...catch捕获下面代码可能出现的异常
  - JSON.parse('{"name":"hello"}')
- 使用try...catch捕获下面代码的异常
  - setTimeout(function(){JSON.parse('{"name":"hello"}')},1000);

回调函数

- Node采用的方法，是将错误对象作为第一个参数，传入回调函数。这样就避免了捕获代码与发生错误的代码不再同一个时间段的问题
- 自己写一个支持异步转换json格式字符串的方法【利用timeout实现就可以了】
  - 要求调用格式为：parse('json字符串',callback(err,obj))

回调函数的设计

- 对于一个函数如果需要定义回调函数，**Node统一规定**：
  - **回调函数一定作为参数的最后一个参数出现**：
    - `function foo(arg1,arg2,callback){}`
  - 回调函数的**第一个参数默认接收错误信息**（便于外界获取调用的错误情况），**第二个参数才是真正的数据**

```
parse('{"foo":"bar"}',function(){err,result})
{
    if(err) throw err;
    console.log(result);
}
```

强调错误优先

- Node统一规定
- 因为之后的操作大多数都是异步的方式
- 异步操作无法通过try-catch捕获异常
- 所以：
  - 错误优先的回调函数，**第一个参数为上一步的错误信息**
  - **通过判断回调函数中的err是否为null来检测异步操作过程是否出现错误**

总结

- Node约定，如果某个函数需要回调函数作为参数，则回调函数是最后一个参数。另外，**回调函数本身的第一个参数，约定为上一步传入的错误对象**
- 传统的错误捕捉机制try...catch对于异步操作行不通
- Node统一规定，一旦异步操作发生错误，就把错误对象传递到回调函数
  - 如果没有发生错误，回调函数的第一个参数就是null
  - 如果不是null，就肯定出错了

ECMAScript 6

使用 ECMAScript 6 编写Node.js应用程序
ECMAScript 5.1 的升级版
向下兼容ECMAScript 5
在原来的基础之上增加了一些东西，或者改进了一些东西

什么是ES6 ？

- http://www.ecma-international.org/ecma-262/6.0/

- ECMAScript是JavaScript语言的国际标准,JavaScript是ECMAScript的实现
- ECMAScript 6 是JavaScript语言的下一代标准，已经在2015年6月正式发布。
- 目标：让JavaScript语言可以用来编写大型的复杂的应用程序,称为企业级开发语言。

ES6支持情况

- ES6已经作为新一代标准发布了，但是各大浏览器对新功能实现支持还需要一段时间
- 对于ES6的支持情况，可以查看下面网站可以了解到不同版本浏览器对ES6的支持情况
- kangax.github.io/es5-compat-table/es6/
- Node.js因为采用了Chrome V8引擎，所以对于ES6的支持非常好，因为运行在服务器端，所以Node开发不用考虑兼容性问题，可以放心大胆的使用了

严格模式：strict mode

- 让JavaScript在更严格的条件下运行。
- 严格模式的目的：
  - 消除JavaScript语法的一些不合理、不严谨之处，减少一些怪异行为
  - 在Node.js中ECMAScript 6 的某些语法只有在严格模式下才支持
- 开启严格模式："use strict";
  - 注意："use strict"必须放在脚本第一行，否则无效

let

- 作用：类似于var，用来声明变量，但是所声明的变量，只在let命令所在的**代码块内有效**
- 不存在变量提升：先声明，后使用，否则报错
- 块级作用域内let声明的变量不受外部的影响，可以定义外层作用域的同名变量
- 不允许在相同作用域内重复声明一个变量
- 应用：for循环计数器

const

- 作用：也是用来声明变量，但是声明的是**常量**。
- 特性：一旦声明，常量的值就不能改变，也是块级作用域
- 注意：
  - 具有块级作用域，但是，**不要在块里面用const**
  - 没有变量提升，先声明后使用，不可以重复声明
  - 使用const只声明不赋值会报错
- const指令指向变量所在的地址，所以对该变量进行属性设置是可以的，赋值会报错

字符串扩展

- includes(str)　表示是否找到了参数字符串
- startsWith(str)　表示参数字符串是否在源字符串的头部
- endsWith(str)　表示参数字符串是否在源字符串的尾部
- repeat(num)　将原字符串重复n次并返回
- 模板字符串``
  - 增强版的字符串：用反引号(`)作为标识
  - 模板字符串中所有的空格和缩进都会被保留
  - 在模板字符串中嵌入变量：${变量名},可以有多个
  - 模板字符串可以是原始的：String.raw`hello world\n`

箭头函数- 一个参数

- ES6允许使用“箭头”（=>）定义函数
- var f = v => v; 等同于：
- var f = function(v){

  return v;

  }

箭头函数- 不需要参数

- `var f = () => 5;`
- 等同于
- `var f = function(){ return 5 };`

箭头函数- 多个参数

```
var sum = (num1, num2) => num1 + num2;
```

等同于

```
var sum = function(num1, num2){
  return num1 + num2;
}
```

箭头函数- 多于一条语句

- 如果箭头函数代码块部分多于一条语句，就要使用大括号将它们括起来。
- `var sum = (num1, num2) => {`
- `  console.log('数字：' + num1);`
- `  return num1 + num2;`
- `}`

箭头函数- 使用场景

- [1,2,3].map(function(x){
  return x * x;
- });
- 箭头函数写法：
- [1,2,3].map(x => x*x);

箭头函数- 注意

- **函数体内的this对象**，绑定定义时所在的对象，而不是使用时所在的对象。
- 不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出异常
- 不可以使用arguments对象，该对象在箭头函数体内不存在

缓冲区处理（二进制数据）

什么是缓冲区
为什么要有缓冲区
缓冲区操作

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
- 而在在大文件操作时必须有缓冲区

文件操作

学习文件以及目录的读写操作

文件操作的几个问题

- 通过fs.readFile(function(err, data){ })读取文件，直接输出data数据
- 在桌面右键新建一个a.txt文件，输入”传智播客”，然后读取并输出该文件内容

二进制数据

- 计算机内部，所有信息最终都表示为一个二进制的字符串。**每一个二进制位(bit)就是 0 和 1 两种状态**
- 八个二进制位就可以组合出 256 种状态，这被称为一个字节。
- 也就是说：一个字节一共可以用来表示 256 种不同的状态，每一个状态对应一个符号,就是 256 个符号，从 00000000 到 11111111

从二进制到文本

- 计算机由许多0和1构成
- 但是人类不是这样
- 有很多编码系统用于将二进制数据转换成文本
- ASCII就是其中一种
- 它是以英文字母为基础并且提供了用来对英语中常用的字母、数字和标点符号编码的一种方法。

文件操作相关模块

- path　　路径字符串操作
- fs　　文件操作
- readline　　按行读取文本文件
- 第三方模块：fs-extra
  - https://www.npmjs.com/package/fs-extra

path模块

- path：处理路径相关字符串
  - basename(p[,ext]) 注意第二个参数用法　获取文件名
  - dirname(p)　获取文件目录
  - extname(p)　获取文件扩展名
  - format(pathObject) 和 parse(pathString)
  - isAbsolute(path)　判断是否是绝对路径
  - join([path1][,path2][,...])　拼接路径字符串
  - normalize(p) 将非标准路径转换为标准路径
  - sep　获取操作系统的文件路径分隔符

同步和异步文件系统调用

- fs模块对文件的几乎所有操作都有同步和异步两种形式
- 例如：readFile()和readFileSync()
- 同步文件系统调用
- 异步文件系统调用
- 同步与异步文件系统调用的区别
  - 异步调用需要回调函数作为额外的参数，通常包含一个错误作为回调函数的第一个参数
  - 异步调用通过判断第一个err对象处理异常，同步必须使用try/catch
  - 同步调用立即执行，会阻塞当前线程
  - 异步调用会进入事件队列，不阻塞后续代码的继续执行，实际的调用直到它被事件循环线程提取出才会执行

文件写入

- 异步文件写入

```
fs.writeFile(file,data[,option],callback(err))
```

- 同步文件写入

```
fs.writeFileSync(file,data,[,option])
```

向文件中追加内容

- 异步追加

```
fs.appendFile(file,data[,options],
callback(err))
```

- 同步追加

```
fs.appendFileSync(file,data[,optio
ns])
```

文件读取

- 异步文件读取

```
fs.readFile(file[,options],callbac
k(err,data))
```

- 同步文件读取

```
fs.readFileSync(file,[,option])
```

其它文件操作任务

- 验证路径是否存在
  - fs.exists(path,callback(exists))
  - fs.existsSync(path) // => 返回布尔类型 exists
- 获取文件信息
  - fs.stat(path,callback(err,stats))
  - fs.statSync(path) // => 返回一个fs.Stats实例【备注】
- 删除文件
  - fs.unlink(path,callback(err))
  - fs.unlinkSync(path)

其它文件操作任务

- 重命名文件或目录
  - fs.rename(oldPath,newPath,callback)
  - fs.renameSync(oldPath,newPath)
- 移动文件
  - fs.rename(oldPath)

目录操作

- 创建一个目录
  - fs.mkdir(path[,model],callback)
  - fs.mkdirSync(path[,model])
- 删除一个空目录
  - fs.rmdir(path,callback)
  - fs.rmdirSync(path)
- 读取一个目录
  - fs.readdir(path,callback(err,files))
  - fs.readdirSync(path) // => 返回files

案例：项目初始化工具

- 需求：
- 在命令台中输入czbk init 文件夹名称
- 自动创建项目的骨架结构
- 骨架结构要求可配置
- 控制台歌词滚动

监视文件

- 监视文件变化：
  - fs.watchFile(filename[, options], listener(curr,prev))
  - options:{persistent,interval}
  - fs.watch(filename[,options][,listener])
- 案例：利用监视文件功能实现一个markdown文件转换器

fs-extra模块介绍

- https://www.npmjs.com/package/fs-extra
- 扩展原生的Node.js的fs模块做不到的功能
- 其实也就是API

文件操作案例

- 项目初始化工具
- markdown文件转换器
- 工资文件翻倍