Node.js

下载配置：https://npm.taobao.org/mirrors/node

> 特点：事件驱动，非阻塞I/O

node.js 模块

> 阿里面试题：用户在浏览器输入 www.taobao.com 直到看到网页，中间发生了什么
>
> (1) 操作系统访问网络的 DNS 服务器，把域名转 IP 地址
>
> (2) 浏览器使用 IP 地址向淘宝发送请求
>
> (3) 淘宝服务器接受请求，并解析请求内容，查找资源，可能是数据库，构建并返回 HTTP 响应消息
>
> (4) 浏览器接受并解析响应消息
>
> (5) 浏览器缓存接受响应内容， 并解析渲染内容

静态和动态网页

> 静态：网页内容任何人任何时间访问都是不变的
>
> HTML/CSS/JS/Flash/视音频

> 动态：网页内容不同人在不同时间访问可能是不同的
>
> DB/JSP/PHP/ASP/Node.JS

> JSP = HTML + JAVA 功能强大可靠，适合大型企业项目（阿里；银行）
>
> 2003 - 2004 淘宝
>
> [php + mysql] -> mysql(Oracle) -> php -> java
>
> -> 服务器 -> 小型机 -> oracle -> { 服务器+Java+OC(mysql) }

> PHP = html + php 简单易用，适合互联网项目
>
> （论坛；Wordpress(https://cn.wordpress.org/)）

> ASP.NET = html + c# 易用，贵

> Node.js = html + js 性能好

node.js 适合项目（io 密集型）	**node.js 不适合项目（cpu 密集型）**
输入输出[查询；添加]	滴滴打车；天气预报

node.js

> node.js 不是 js，是一种运行在 **服务器** 端开发平台（开发语言用 js）
>
> 历史上第一种可以通吃前后台的语言
>
> 淘宝镜像：https://npm.taobao.org/mirrors/node

node.js 运行模式

(1) 交互模式 —— 用于一般测试

输入一行，代码执行一行

> 命令行输入
>
> 进入：node + enter
>
> 退出：ctrl + c + c

(2) 脚本模式 —— 用于正式项目中

把要执行的所有语句编写在一个文本文件中（后缀名任意，没有也行），

一次性提交 node 解释器执行

> node x.js + 回车

解释器默认位置：C:\Program Files\nodejs

自学新语言

如何自学一门新语言

> (1) 了解背景
>
> (2) 搭建开发环境，编写 hello world
>
> (3) 数据类型

前端 js 数据类型	node.js 数据类型
1. 原始 number; string; boolean; null; undefined 2. 引用/对象类型 ES 对象：Math;Date;RegExp;Object; ... *BOM/DOM 对象：* window;document;screen;event ... 自定义：{}	1. 原始 number; string; boolean; null; undefined; 2. 引用 / 对象类型 ES 对象：Math; Date; Object ... 自定义：{} Node.js 提供了十几万个第三方模块 www.npmjs.com

(4) 变量和常量

```
var age = 19;
const PI = 3.14
```

(5) 运算符

逻辑运算符

比较运算符

算术运算符

三目运算符

位运算符

赋值运算符

特殊　　typeof / instanceof

(6) 逻辑结构

循环结构：while; do...while(); for(;;); for(..in..); for(..of..)

选择结构：if..else switch..case

(7) 通用小程序

九九乘法表，100 以内的质数，数组排序 ...

(8) 函数和对象

(9) 常用组件，第三方工具，框架

(10) 实际小项目

(11) 撸手册

Node.js 特有 -- 模块

一个 web 项目的功能可以分为很多不同 “模块”，如商品模块，订单模块，支付模块 。。。

node.js 中按功能不同，可以把函数，对象保存在不同的文件或目录下，这些文件或目录称为“module”

> node.js 中每个模块都是一个独立的构造函数，解析器会自动为每个 .js 文件添加如下代码：

```
(function(exports, require, module, filename, dirname){
  exports: {} 用于声明向外部导出自己的成员
  require: fn 用于导入其他模块，创建指定模块对象
  module: 指代当前模块对象
  __filename: 当前文件绝对路径
  __dirname: 当前文件目录绝对路径
  // 自己的代码保存在这里
})
```

每个模块可以使用require()函数引入另一个模块 —— 底层创建指定模块的对象实例,

`require("./模块文件名");`

每个模块使用exports对象对外导出/公开一些自己的内容成员供其它的模块使用

`exports.成员名 = 成员值;`

Node.js 的模块中 exports 和 module.exports 对象区别是什么？
二者都可以用于向外界导出自己的内部成员
module 变量指代当前模块对象，真正导出的数据的是 module.exports
Node.js 底层代码 exports = module.exports
=> 若只是导出 属性 则 exports = module.exports
=> 若导出完整对象 module.exports

模块分类

(1) Node.js 官方提供的模块 -- 安装在解释器中

require("模块名称");

(2) 第三方模块

(3) 用户自定义模块

module.exports.x = require("模块文件名");

官方提供的模块

—— Global

> 该模块是官方内嵌模块，可以直接使用，无需 require("global");

属性	描述
exports	用于声明向外部导出自己的成员
module	指代当前模块对象
require	用于导入其他模块，创建指定模块对象
__filename	当前文件绝对路径
__dirname	当前文件目录绝对路径
console	指当前控制台对象 注意：该对象与 Chrome 中 console 不同 console.log/dir/time/timeEnd/asserr ...
setInterval(fn, time);	周期性定时器
setTimeout(fn, time);	一次性定时器
setImmediate(fn);	立即执行 == setTimeout(fn,0);

node.js 模块分类

(1) 官方提供的模块

(2) 第三方模块，单独下载 www.npmjs.com

(3) 自定义模块

- 文件模块：创建一个 js 文件，如 m3.js 导出需要的公开数据，其他模块 require("./m3"); 模块
- 目录模块：
  - 方式一：
    - 创建一个目录，假设名为 m4，
    - 其中创建 index.js 导出需要公开的数据，其他模块引用 require("./m4");
  - 方式二：
    - 创建一个目录,假设名为 m5,其中创建名为 5.js
    - 创建 package.json， main属性指定启动文件 5.js
  - 方式三：
    - 创建一个目录，必须名为 `node_modules` ，
    - 其中创建一个目录，假设 m6，
    - m6中创建 package.json 描述文件，
    - m6中声明 main 属性，指定默认执行 js 文件，如 6.js，
    - 6.js中导出公开数据，其他模块 require("m6")

node.js 特性

—— **事件驱动** ； **非阻塞 IO**

> 阻塞 IO：PHP, JSP —— 有序，效率差
>
> 非阻塞 IO ：Node.js —— 无序，效率高

npm

npm(Node Package Manager)

> node.js 的第三方模块 / 包管理器，可以用于下载；更新；删除；维护包依赖关系的工具
>
> npm 工具默认从 www.npmjs.com 网站下载所需的第三方模块包

> 使用 npm 工具下载一个新的软件包
>
> npm install *包名* // 下载
>
> npm uninstall *包名* // 卸载

官方模块 -- querystring

querystring 模块用于处理 HTTP 请求中查询字符串

> var obj = qs.parse(str); 将 查询字符串 转换为 对象
>
> var str = qs.stringify(obj); 把 js对象 转换为 查询字符串

官方模块 -- url

url 模块用于分解一个 HTTP 请求地址，获取其中各个不同的部分

> var obj = url.parse(str); 把一个 URL 字符串解析为一个对象
>
> var obj = url.parse(str, true); 把一个 URL 字符串解析为一个对象，并把其中 **查询字符串** 转换为对象

官方模块 -- buffer(缓冲区)

buffer：缓冲区，本质是一块内存区域，用于暂时存储数据（数字；字符串；图片；音视频 ... ）

> - 创建一个缓冲区 1024 字节(byte)
>
>   var buffer = Buffer.alloc(1024);
> - 创建一个数字数组缓冲区
>
>   var buffer1 = Buffer.from([1,2,3,4]);
> - 创建一个字符串缓冲区
>
>   var buffer2 = Buffer.from("哈撒给");

> - 将缓冲区转换为字符串
>
>   var buffer3 = Buffer.toString();
>
> 计算机常用容量单位
>
> 8 bit = 1 byte
>
> 1024b = 1kb
>
> 1024mb = 1gb
>
> 1024gb = 1tb
>
> 1024tb = 1pb as 21

官方模块 -- fs ☆

fs 模块提供了对文件系统中的文件 / 目录进行增删改查，读写功能

> **阻塞方案**
>
> - 读取文件内容
>
>   var data = fs.readFileSync(*filename*);
> - 向文件中写内容（覆盖原内容）
>
>   fs.writeFileSync(*filename*, *content*);
> - 向文件中追加内容
>
>   fs.appendFileSync(*filename*, *content*);
>
> **非阻塞方案**
>
> - 读取文件内容
>
>   var data = fs.readFile(*filename*, (err, data)=>{
>
>   // 回调函数，当文件读取结束执行
>
>   // data 内容；err 错误
>
>   });
> - 向文件中写内容（覆盖原内容）
>
>   fs.writeFile(*filename*, *content*, (err)=>{
>
>   // 当文件写完成后，调用函数
>
>   });
> - 向文件中 **追加** 内容
>
>   fs.appendFile(*filename*, *content*, (err)=>{
>
>   // 当文件追加完，执行函数
>
>   });

官方模块 -- http ☆

HTTP 模块可用于编写基于 HTTP 协议客户端程序（浏览器）或服务器程序（如 Apache）

> 用 http 模块编写一个 web 服务器
>
> (1) 创建一个服务器对象
>
> var server = http.createServer();
>
> (2) 绑定监听端口
>
> server.listen(*port*);
>
> // 3306 mysql / https 443 / www/ 80 (端口 1 ~ 65535)
>
> (3) 注册事件，客户端请求事件
>
> server.on("request", (req,res) => {});
>
> "request"：事件 request 客户请求
>
> req：请求对象（客户端）
>
> res：响应对象（服务器）
>
> (4) 向客户端返回消息
>
> res.write(); res.end();
>
> > 指定响应给客户端字符集编码格式
> >
> > res.setHeader("Content-Type", "text/html;charset=UTF-8");

第三方模块 -- mysql

> 为了精简 node.js 解释器，官方没有提供访问任何数据库相关模块，
>
> 必须使用 npm 工具下载第三方模块 www.npmjs.com

使用步骤：

> (1) 创建到数据库服务器连接
>
> ```
> const mysql = require("mysql");
> var conn = mysql.createConnection({
>     host: "",
>     user: "",
>     password: "",
>     port: "",
>     database: ""
> });
> ```
>
> (2) 发送 sql 语句给服务器执行

```
conn.query("SQL语句", (err, result)=>{});
```

(3) 关闭连接

```
conn.end();
```

mysql 模块 -- 网络安全 -- SQL注入

SQL 注入：利用 sql 语句常见规则，将危险代码加入 SQL语句，对系统造成破坏

比如将 DROP TABLE / SELECT 注入

解决：

Node.js 使用占位符技术

```
var sql = "SELECT COUNT(id) as c FROM xz_admin WHERE uname=? AND upwd=md5(?)";
conn.query(sql, [uname, upwd], (err, res)=>{});
```

连接池

为了提高系统效率，mysql 创建连接池（池，提高效率）

开发流程：

(1) 创建连接池

```
const mysql = require("mysql");
var pool = mysql.createPool({
  host:;user:;password:;port:;database:;
  connectionLimit: 5
});
```

(2) 从连接池租一个连接发送并且发送 sql语句。自动回收连接

pool.query(sql, [], (err, result)=>{})

js对象 转换 json字符串 `JSON.stringify(str);`

var json = JSON.stringify(rows);

第三方模块 -- express

使用官方提供HTTP模块可以创建一个web服务器

但是此模非常底层,要处理各种情形,比较繁琐

推荐使用 http 模块进一步封装简化模块

-- express 第三方模块

使用步骤

(1) 加载模块 express

```
const express = require("express");
```

(2) 创建 express 对象

```
var app = express();
```

(3) 创建服务器对象

```
const http = require("http");
var server = http.createServer(app);
```

(4) 绑定监听端口 3000

```
server.listen(3000);
```

EX: 客户端 /add.html GET

```
app.get("/add.html", (req, res)=>{
    // 以前解析 setHeader + fs.read + write + end
    res.sendFile(__dirname + "/add.html");
    // 向客户端发送 html字符串
    res.send(str);

});
```

常见错误：

TypeError: path must be absolute or specify root to res.sendFile

sendFile 发送文件必须使用绝对路径

请求方法：

请求方式	描述
GET	客户端想 **获取** 服务器资源（html；json；text；jpg）；
POST	客户端 **上传/添加** 指定数据到服务器；（相关数据在请求主体中）
DELETE	客户端想 **删除** 服务器上指定数据；
PUT	客户端想 **更新** 服务器上指定数据；

EX:	描述
GET /userlist	获取用户列表
GET /userlist?pno=2	获取第2页用户列表
GET /user?uid=10	获取编号为10用户信息
POST /user?uname=tom&upwd=123	客户端添加一条数据在服务器
PUT /user?upwd=123&uid=3	客户端想更新服务器指定用户密码
DELETE /user?uid=9	客户想删除编号为9用户
DELETE /user/9	客户想删除编号为9用户

浏览器如何发起 GET 请求

> 地址栏输入 URL; AJAX-GET; form表单; a超链接; js 跳转; src属性; href;

浏览器如何发起 POST 请求

> AJAX-POST; form表单;

浏览器如何发起 PUT 请求

> AJAX-PUT;

浏览器如何发起 DELETE 请求

> AJAX-DELETE;

参数

GET 接受参数

- 接受查询字符串

  GET /uer?uid=3&loc=bj

  app.get("/user", (req, res)=>{

  // express 为每个request对象添加了属性 query

  req.query.uid;

  req.query.loc;

  });
- 接受请求参数

  GET /book/js/60

  express 为每个参数自动创建变量 :name

  创建 name 保存 js；创建 price 保存 60；

  express 针对请求参数创建属性 params

  app.get("/book/:name/:price", (req, res)=>{

```
        req.params.name;
        req.params.price;
    });
```

中间件

> express是一个自身功能极简的框架，如果需要添加新功能就要创建中间件，添加到 express 中间件

> 中间件(Middleware) 是一个（特殊）函数，
>
> 它可以访问请求对象和响应对象(req, res)，控制请求响应流程，控制流程 next 函数完成

分类：

- 应用级中间件
- 路由级中间件
- 内置中间件
- 第三方中间件

应用级中间件

路由 = 请求方式 + 请求地址 + 处理函数　ex: app.get("/user", (req, res)=>{})

```
app.use(url, (req, res, next) => {
  // 中间件要执行的代码
  next(); // 调用下一个中间件或路由，放行
});
// url : 触发中间件工作的请求地址
```

内置中间件 static

> express 3 提供了十几个内置中间件
>
> express 4 提供了一个内置中间件
>
> > static 静态资源目录中间件
> >
> > 作用：指定静态资源的目录，当用户请求该目录所有资源，express 读取文件并返回文件内容
> >
> > ```
> > app.use(express.static("public"));
> > ```

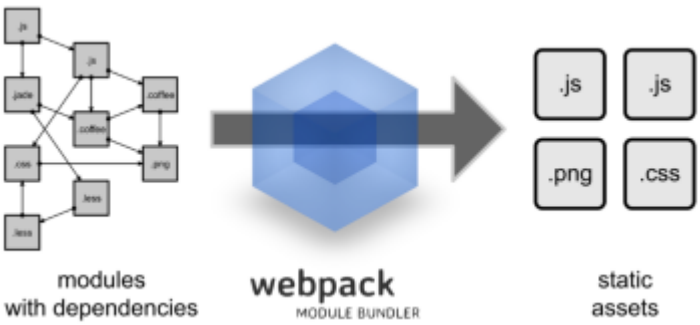
路由级中间件

> 作用：将应用程序中功能分派不同文件中，减少 app.js 内容，解决命名冲突
>
> EX: app.js 所有功能在此处理，效率低，不能多人协作
>
> 解决：app.js = user.js + product.js + order.js + cart.js

```
// user.js 创建路由对象 处理各种请求 /list /del /update /search
app.use("/user", 路由对象);
app.use("/product", 路由对象);
app.use("/order", 路由对象);
```

Webpack

> 安装部署工具(Vue / NG / React / ...)
>
> webpack 打包处理工具(gulp);
>
> webpack 一款处理加载器打包工具，他能把各种资源(js/jsx/less/sass/css/png)打包到一个或多个文件



webpack 安装和配置

> 全局安装 webpack
>
> ```
> npm i webpack@3.3.0 -g
> ```
>
> 本地目录安装 webpack
>
> ```
> npm i webpack@3.3.0 -save-dev
> ```

案例一：只用 webpack 打包一个 .js 文件

```
#创建目录 e:/webpackdemo/01
#目标: 01/test.js  打包  bundle.js
#打包语法: webpack 源文件  目标文件
  webpack test.js  bundle.js   11:50
#index.html 引用bundle.js
```

案例二:使用webpack打包一个文件(引其它 js)

```
#02/ test.js word.js
#打包:
  webpack test.js  bundle.js
```

案例三:使用webpack配置文件

```
03/test.js
03/webpack.config.js
module.exports = {
  entry:"./test.js",              #入口文件
  output:[filename:"bundle.js"]   #输出文件
}
打包命令  webpack
```

案例四:使用webpack配置文件 打包多个文件

```
04/main1.js  bundle1.js
04/main2.js  bundle2.js
打包命令  webpack
```

案例五:使用webpack配置文件打包--添加参数

```
webpack --progress         打包时显示进度条
webpack --display-modules  打包时显示加载模块
webpack --watch           热部署(监听代码改动并且自动打包)
```

案例六:使用package.json 打包程序

```
//06/test.js
//06/webpack.config.js
//创建package.json模块描述文件
  npm init
//添加项目
//运行命令   npm run dev
```

案例七:打包 css程序

```
//单独安装第三方模块 加载器 {css-loader;style-loader}
  npmi css-loader style-loader --save-dev
//创建index.cssindex.html
//创建test.js 引入 index.css
//打包
  webpack.config.js
```

案例八：打包图片

```
npm i url-loader file-loader --save-dev
webpack.config.js
webpack
```