node.js 介绍

node.js 初探

node.js 是一个 JS 的服务端运行环境，简单的来说，他是在 JS 语言规范的基础上，封装了一些服务端的运行时对象，让我们能够简单实现非常多的业务功能。

如果我们只使用 JS 的话，实际上只是能进行一些简单的逻辑运算。**node.js 就是基于 JS 语法增加与操作系统之间的交互。**

node.js 的安装

我们可以使用多种方式来安装 node.js，node.js 本质上也是一种软件，我们可以使用直接下载二进制安装文件安装，通过系统包管理进行安装或者通过源码自行编译均可。

一般来讲，对于个人开发的电脑，我们推荐直接通过 node.js 官网的二进制安装文件来安装。对于打包上线的一些 node.js 环境，也可以通过二进制编译的形式来安装。

安装成功之后，我们的 node 命令就会自动加入我们的系统环境变量 path 中，我们就能直接在全局任意位置，使用 node 命令访问到我们刚才安装的 **node 可执行命令行工具**。

node.js 版本切换

在个人电脑上，我们可以安装一些工具，对 node.js 版本进行切换，例如 nvm 和 n。

nvm 的全称就是 node version manager，意思就是能够管理 node 版本的一个工具，它提供了一种直接通过 shell 执行的方式来进行安装。简单来说，就是通过将多个 node 版本安装在指定路径，然后通过 nvm 命令切换时，就会切换我们环境变量中 node 命令指定的实际执行的软件路径。

```
curl -o-
https://raw.githubusercontent.com/nvm-
sh/nvm/v0.35.3/install.sh | bash
```

安装成功之后，我们就能在当前的操作系统中使用多个 node.js 版本。

node.js 的包管理工具 npm

我们对 npm 应该都比较熟悉了，它是 node.js 内置的一款工具，目的在于安装和发布符合 node.js 标准的模块，从而实现社区共建的目的繁荣整个社区。

npx 是 npm@5 之后新增的一个命令，它使得我们可以在不安装模块到当前环境的前提下，使用一些 cli 功能。

```
# 此时全局安装了 create-react-app
npm i -g create-react-app
create-react-app some-repo

# 此时无论是项目中还是全局都没有安装 create-react-app (但实际上是安装了的，但表现确实像没有安装)
npx create-react-app some-repo
```

node.js 的底层依赖

node.js 的主要依赖子模块有以下内容：

- V8 引擎：主要是 JS 语法的解析，有了它才能识别 JS 语法
- libuv: c 语言实现的一个高性能异步非阻塞 IO 库，用来实现 node.js 的事件循环
- http-parser/llhttp: 底层处理 http 请求，处理报文，解析请求包等内容
- openssl: 处理加密算法，各种框架运用广泛

- zlib: 处理压缩等内容

node.js 常见内置模块

node.js 中最主要的内容，就是实现了一套 CommonJS 的模块化规范，以及内置了一些常见的模块。

- fs: 文件系统，能够读取写入当前安装系统环境中硬盘的数据
- path: 路径系统，能够处理路径之间的问题
- crypto: 加密相关模块，能够以标准的加密方式对我们的内容进行加解密
- dns: 处理 dns 相关内容，例如我们可以设置 dns 服务器等等
- http: 设置一个 http 服务器，发送 http 请求，监听响应等等
- readline: 读取 stdin 的一行内容，可以读取、增加、删除我们命令行中的内容
- os: 操作系统层面的一些 api，例如告诉你当前系统类型及一些参数
- vm: 一个专门处理沙箱的虚拟机模块，底层主要来调用 v8 相关 api 进行代码解析。

node.js CommonJS 详解及源码解析

V8 引擎：https://github.com/v8/v8 / https://chromium.googlesource.com/v8/v8.git

引擎只是解析层面，具体的上层还有许多具体环境的封装，今天我们主要就是解析一下 node.js 源码中，引擎上层还需要做很多工作，今天我们主要就是讲解一个，使用 CommonJS 模块化时，node.js 源码是如何去做的。

node.js 周边工具简介及解析

node.js debug & 内存泄漏

对于浏览器的 JS 代码来说，我们可以通过断点进行分步调试，每一步打印当前上下文中的变量结果，来定位具体问题出现在哪一步。

我们可以借助 VSC 或者自行打断点的形式，来进行分步 node.js 调试。

对于 JS 内存泄漏，我们也可以使用同样的道理，借助工具，打印每次的内存快照，对比得出代码中的问题。

另一种 JS 解析引擎 quickjs

quickjs 是一个 JS 的解析引擎，轻量代码量也不大，与之功能类似的就是 V8 引擎。

他最大的特点就是，非常非常轻量，这点从源码中也能提现，事实上并没有太多的代码，它的主要特点和优势：

- 轻量而且易于嵌入：只需几个C文件，没有外部依赖，一个x86下的简单的“hello world”程序只要180 KiB。
- 具有极低启动时间的快速解释器： 在一台单核的台式PC上，大约在100秒内运行ECMAScript 测试套件156000次。运行时实例的完整生命周期在不到300微秒的时间内完成。
- 几乎完整实现ES2019支持，包括： 模块，异步生成器和和完整Annex B支持 (传统的Web兼容性)。许多ES2020中带来的特性也依然会被支持。
- 通过100%的ECMAScript Test Suite测试。
- 可以将Javascript源编译为没有外部依赖的可执行文件。

另一类 JS 运行时服务端环境 deno

deno 是一类类似于 node.js 的 JS 运行时环境，同时他也是由 node.js 之父一手打造出来的，他和 node.js 比有什么区别呢?

相同点：

- deno 也是基于 V8，上层封装一些系统级别的调用
- 我们的 deno 应用也可以使用 JS 开发

不同点：

- deno 基于 rust 和 typescript 开发一些上层模块，所以我们可以直接在 deno 应用中书写 ts
- deno 支持从 url 加载模块，同时支持 top level await 等特性

deno 在今年 5 月份会发布 1.0 版本，大家拭目以待它后续的表现吧！