



黑马程序员™
www.itheima.com

传智播客旗下
高端IT教育品牌

初识Node.js与内置模块

目录 Contents

- ◆ 初识 Node.js
- ◆ fs 文件系统模块
- ◆ path 路径模块
- ◆ http 模块

1. 初识 Node.js

1.1 回顾与思考

1. 已经掌握了哪些技术



HTML



CSS

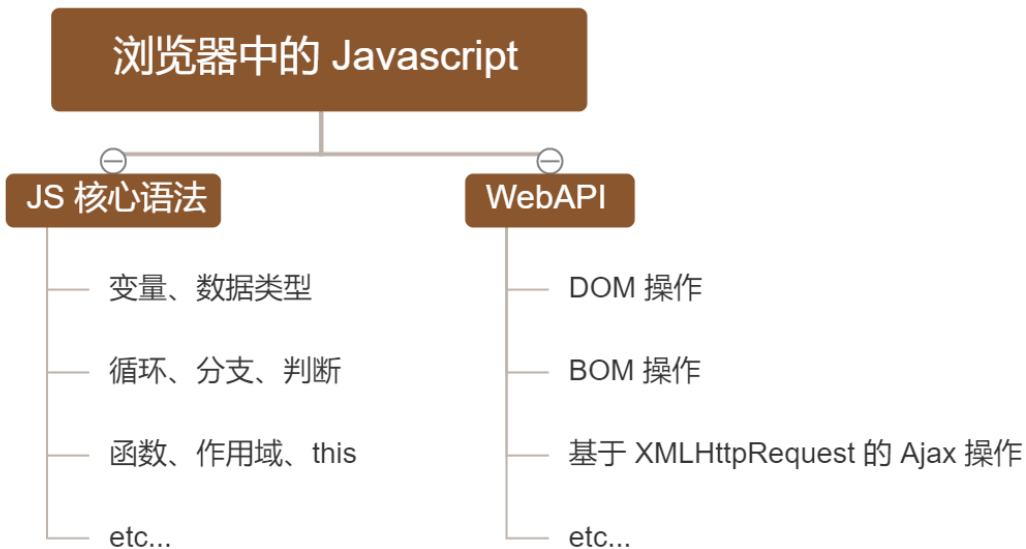


Javascript

1. 初识 Node.js

1.1 回顾与思考

2. 浏览器中的 JavaScript 的组成部分



■ 1. 初识 Node.js

1.1 回顾与思考

3. 思考：为什么 JavaScript 可以在浏览器中被执行



不同的浏览器使用不同的 JavaScript 解析引擎：

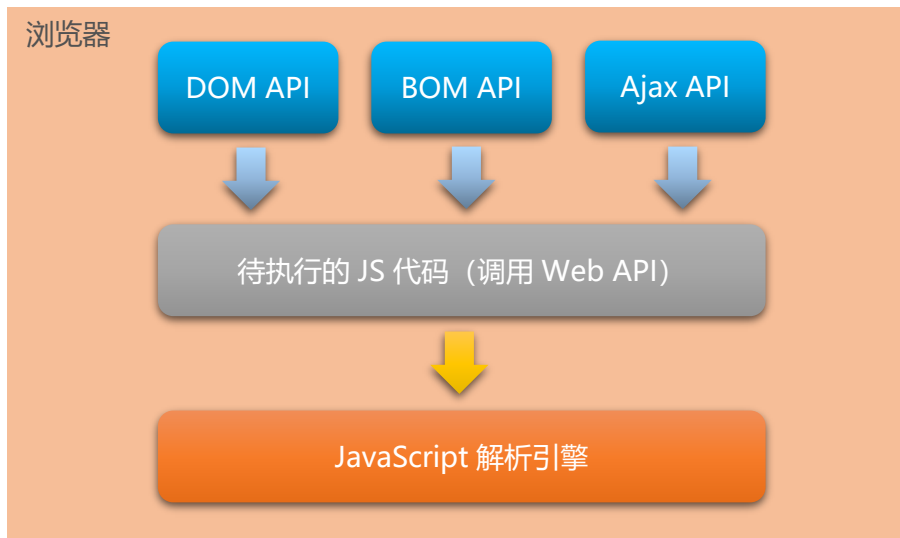
- Chrome 浏览器 => V8
- Firefox 浏览器 => OdinMonkey (奥丁猴)
- Safari 浏览器 => JSCore
- IE 浏览器 => Chakra (查克拉)
- etc...

其中，Chrome 浏览器的 V8 解析引擎性能最好！

1. 初识 Node.js

1.1 回顾与思考

4. 思考：为什么 JavaScript 可以操作 DOM 和 BOM



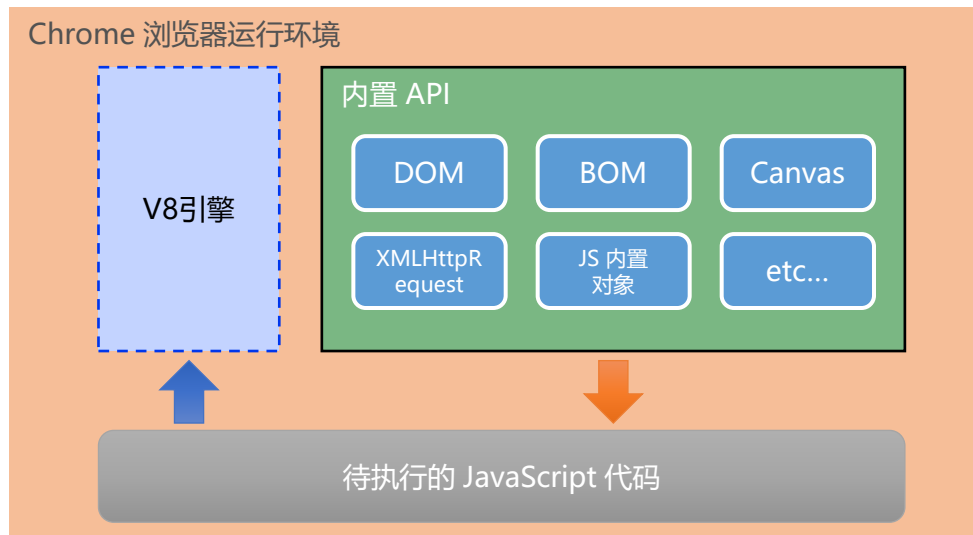
每个浏览器都**内置了** DOM、BOM 这样的 API 函数，因此，浏览器中的 JavaScript 才可以调用它们。

1. 初识 Node.js

1.1 回顾与思考

5. 浏览器中的 JavaScript 运行环境

运行环境是指**代码正常运行所需的必要环境**。



总结：

- ① V8 引擎负责解析和执行 JavaScript 代码。
- ② 内置 API 是由**运行环境**提供的特殊接口，**只能在所属的运行环境中被调用**。

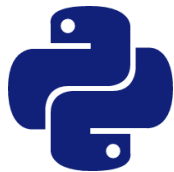
1. 初识 Node.js

1.1 回顾与思考

6. 思考：JavaScript 能否做后端开发



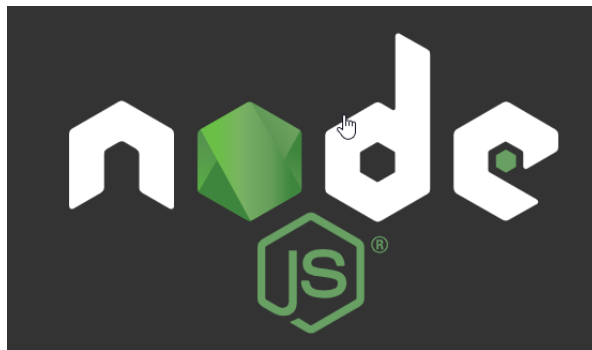
Java



Python



PHP



Node.js



1. 初识 Node.js

1.2 Node.js 简介

1. 什么是 Node.js

Node.js® is a **JavaScript runtime** built on Chrome's V8 JavaScript engine.

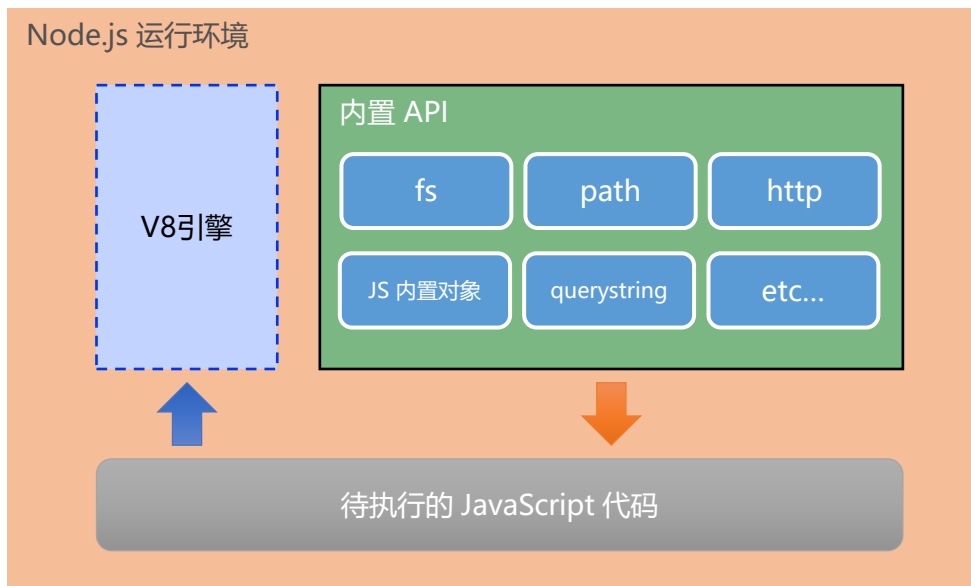
Node.js 是一个基于 Chrome V8 引擎的 **JavaScript 运行环境**。

Node.js 的官网地址: <https://nodejs.org/zh-cn/>

1. 初识 Node.js

1.2 Node.js 简介

2. Node.js 中的 JavaScript 运行环境



注意:

- ① 浏览器是 JavaScript 的前端运行环境。
- ② Node.js 是 JavaScript 的后端运行环境。
- ③ Node.js 中无法调用 DOM 和 BOM 等浏览器内置 API。



1. 初识 Node.js

1.2 Node.js 简介

3. Node.js 可以做什么

Node.js 作为一个 JavaScript 的运行环境，仅仅提供了基础的功能和 API。然而，基于 Node.js 提供的这些基础能力，很多强大的工具和框架如雨后春笋，层出不穷，所以学会了 Node.js，可以让前端程序员胜任更多的工作和岗位：

- ① 基于 Express 框架 (<http://www.expressjs.com.cn/>)，可以快速构建 Web 应用
- ② 基于 Electron 框架 (<https://electronjs.org/>)，可以构建跨平台的桌面应用
- ③ 基于 restify 框架 (<http://restify.com/>)，可以快速构建 API 接口项目
- ④ 读写和操作数据库、创建实用的命令行工具辅助前端开发、etc...

总之：Node.js 是**大前端时代**的“大宝剑”，有了 Node.js 这个超级 buff 的加持，前端程序员的**行业竞争力**会越来越强！

■ 1. 初识 Node.js

1.2 Node.js 简介

4. Node.js 好学吗

好学!

会 JavaScript, 就能学会 Node.js! ! !

■ 1. 初识 Node.js

1.2 Node.js 简介

4. Node.js 怎么学

浏览器中的 JavaScript 学习路径：

JavaScript 基础语法 + 浏览器内置 API (DOM + BOM) + 第三方库 (jQuery、art-template 等)

Node.js 的学习路径：

JavaScript 基础语法 + Node.js 内置 API 模块 (fs、path、http等) + 第三方 API 模块 (express、mysql 等)

1. 初识 Node.js

1.3 Node.js 环境的安装

如果希望通过 Node.js 来运行 Javascript 代码，则必须在计算机上安装 Node.js 环境才行。

安装包可以从 Node.js 的官网首页直接下载，进入到 Node.js 的官网首页 (<https://nodejs.org/en/>)，点击绿色的按钮，下载所需的版本后，双击直接安装即可。

Download for Windows (x64)

12.16.1 LTS

Recommended For Most Users

13.11.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)



1. 初识 Node.js

1.3 Node.js 环境的安装

1. 区分 LTS 版本和 Current 版本的不同

- ① LTS 为长期稳定版，对于追求稳定性的企业级项目来说，推荐安装 LTS 版本的 Node.js。
- ② Current 为新特性尝鲜版，对热衷于尝试新特性的用户来说，推荐安装 Current 版本的 Node.js。但是，Current 版本中可能存在隐藏的 Bug 或安全性漏洞，因此不推荐在企业级项目中使用 Current 版本的 Node.js。



1. 初识 Node.js

1.3 Node.js 环境的安装

2. 查看已安装的 Node.js 的版本号

打开终端，在终端输入命令 `node -v` 后，按下回车键，即可查看已安装的 Node.js 的版本号。

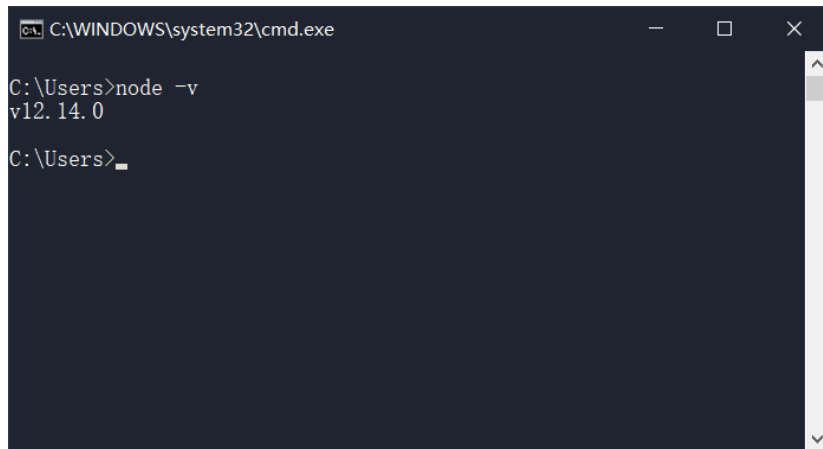
Windows 系统快速打开终端的方式：

使用快捷键（**Windows徽标键** + **R**）打开运行面板，输入 `cmd` 后直接回车，即可打开终端。

■ 1. 初识 Node.js

1.3 Node.js 环境的安装

3. 什么是终端



```
C:\WINDOWS\system32\cmd.exe
C:\Users>node -v
v12.14.0
C:\Users>_
```

终端（英文：Terminal）是专门为开发人员设计的，**用于实现人机交互**的一种方式。

作为一名合格的程序员，我们有必要识记一些**常用的终端命令**，来辅助我们更好的操作与使用计算机。



1. 初识 Node.js

1.4 在 Node.js 环境中执行 JavaScript 代码

- ① 打开终端
- ② 输入 `node` 要执行的js文件的路径



1. 初识 Node.js

1.4 在 Node.js 环境中执行 JavaScript 代码

1. 终端中的快捷键

在 Windows 的 powershell 或 cmd 终端中，我们可以通过如下快捷键，来提高终端的操作效率：

- ① 使用 **↑** 键，可以快速定位到上一次执行的命令
- ② 使用 **tab** 键，能够快速补全路径
- ③ 使用 **esc** 键，能够快速清空当前已输入的命令
- ④ 输入 **cls** 命令，可以清空终端

目录 Contents

- ◆ 初识 Node.js
- ◆ fs 文件系统模块
- ◆ path 路径模块
- ◆ http 模块

2. fs 文件系统模块

2.1 什么是 fs 文件系统模块

fs 模块是 Node.js 官方提供的、用来操作文件的模块。它提供了一系列的方法和属性，用来满足用户对文件的操作需求。

例如：

- `fs.readFile()` 方法，用来**读取**指定文件中的内容
- `fs.writeFile()` 方法，用来向指定的文件中**写入**内容

如果要在 JavaScript 代码中，使用 fs 模块来操作文件，则需要使用如下的方式先导入它：

```
1 const fs = require('fs')
```

2. fs 文件系统模块

2.2 读取指定文件中的内容

1. fs.readFile() 的语法格式

使用 fs.readFile() 方法，可以读取指定文件中的内容，语法格式如下：

```
1 fs.readFile(path[, options], callback)
```

参数解读：

- 参数1：必选参数，字符串，表示文件的路径。
- 参数2：可选参数，表示以什么编码格式来读取文件。
- 参数3：必选参数，文件读取完成后，通过回调函数拿到读取的结果。

2. fs 文件系统模块

2.2 读取指定文件中的内容

2. fs.readFile() 的示例代码

以 utf8 的编码格式，读取指定文件的内容，并打印 err 和 dataStr 的值：

```
1 const fs = require('fs')
2 fs.readFile('./files/11.txt', 'utf8', function(err, dataStr) {
3   console.log(err)
4   console.log('-----')
5   console.log(dataStr)
6 })
```

2. fs 文件系统模块

2.2 读取指定文件中的内容

3. 判断文件是否读取成功

可以判断 err 对象是否为 null，从而知晓文件读取的结果：

```
1 const fs = require('fs')
2 fs.readFile('./files/1.txt', 'utf8', function(err, result) {
3   if (err) {
4     return console.log('文件读取失败! ' + err.message)
5   }
6   console.log('文件读取成功, 内容是: ' + result)
7 })
```


2. fs 文件系统模块

2.3 向指定的文件中写入内容

1. fs.writeFile() 的语法格式

使用 fs.writeFile() 方法，可以向指定的文件中写入内容，语法格式如下：

```
1 fs.writeFile(file, data[, options], callback)
```

参数解读：

- 参数1：必选参数，需要指定一个文件路径的字符串，表示文件的存放路径。
- 参数2：必选参数，表示要写入的内容。
- 参数3：可选参数，表示以什么格式写入文件内容，默认值是 utf8。
- 参数4：必选参数，文件写入完成后的回调函数。

2. fs 文件系统模块

2.3 向指定的文件中写入内容

2. fs.writeFile() 的示例代码

向指定的文件路径中，写入文件内容：

```
1 const fs = require('fs')
2 fs.writeFile('./files/2.txt', 'Hello Node.js!', function(err) {
3   console.log(err)
4 })
```

2. fs 文件系统模块

2.3 向指定的文件中写入内容

3. 判断文件是否写入成功

可以判断 err 对象是否为 null，从而知晓文件写入的结果：

```
1 const fs = require('fs')
2 fs.writeFile('F:/files/2.txt', 'Hello Node.js!', function(err) {
3   if (err) {
4     return console.log('文件写入失败! ' + err.message)
5   }
6   console.log('文件写入成功! ')
7 })
```

2. fs 文件系统模块

2.5 练习 - 考试成绩整理

使用 fs 文件系统模块，将素材目录下成绩.txt文件中的考试数据，整理到成绩-ok.txt文件中。

整理前，成绩.txt文件中的数据格式如下：

```
1 小红=99 小白=100 小黄=70 小黑=66 小绿=88
```

整理完成之后，希望得到的成绩-ok.txt文件中的数据格式如下：

```
1 小红：99
2 小白：100
3 小黄：70
4 小黑：66
5 小绿：88
```

2. fs 文件系统模块

2.5 练习 - 考试成绩整理

核心实现步骤

- ① 导入需要的 fs 文件系统模块
- ② 使用 `fs.readFile()` 方法，读取素材目录下的 `成绩.txt` 文件
- ③ 判断文件是否读取失败
- ④ 文件读取成功后，处理成绩数据
- ⑤ 将处理完成的成绩数据，调用 `fs.writeFile()` 方法，写入到新文件 `成绩-ok.txt` 中

2. fs 文件系统模块

2.6 fs 模块 - 路径动态拼接的问题

在使用 fs 模块操作文件时，如果提供的操作路径是以 `./` 或 `../` 开头的**相对路径**时，很容易出现路径动态拼接错误的问题。

原因：代码在运行的时候，**会以执行 node 命令时所处的目录**，动态拼接出被操作文件的完整路径。

解决方案：在使用 fs 模块操作文件时，**直接提供完整的路径**，不要提供 `./` 或 `../` 开头的相对路径，从而防止路径动态拼接的问题。

```
1 // 不要使用 ./ 或 ../ 这样的相对路径
2 fs.readFile('./files/1.txt', 'utf8', function(err, dataStr) {
3   if (err) return console.log('读取文件失败! ' + err.message)
4   console.log(dataStr)
5 })
6
7 // __dirname 表示当前文件所处的目录
8 fs.readFile(__dirname + '/files/1.txt', 'utf8', function(err, dataStr) {
9   if (err) return console.log('读取文件失败! ' + err.message)
10  console.log(dataStr)
11 })
```

目录 Contents

- ◆ 初识 Node.js
- ◆ fs 文件系统模块
- ◆ path 路径模块
- ◆ http 模块

3. path 路径模块

3.1 什么是 path 路径模块

path 模块是 Node.js 官方提供的、用来**处理路径**的模块。它提供了一系列的方法和属性，用来满足用户对路径的处理需求。

例如：

- `path.join()` 方法，用来**将多个路径片段拼接成一个完整的路径字符串**
- `path.basename()` 方法，用来从路径字符串中，将文件名解析出来

如果要在 JavaScript 代码中，使用 path 模块来处理路径，则需要使用如下的方式先导入它：

```
1 const path = require('path')
```


3. path 路径模块

3.2 路径拼接

1. path.join() 的语法格式

使用 path.join() 方法，可以把多个路径片段拼接为完整的路径字符串，语法格式如下：

```
1 path.join([...paths])
```

参数解读：

- ...paths <string> 路径片段的序列
- 返回值: <string>

3. path 路径模块

3.2 路径拼接

2. path.join() 的代码示例

使用 path.join() 方法，可以把多个路径片段拼接为完整的路径字符串：

```
1 const pathStr = path.join('/a', '/b/c', '../', './d', 'e')
2 console.log(pathStr) // 输出 \a\b\d\e
3
4 const pathStr2 = path.join(__dirname, './files/1.txt')
5 console.log(pathStr2) // 输出 当前文件所处目录\files\1.txt
```

注意：今后凡是涉及到路径拼接的操作，都要使用 path.join() 方法进行处理。不要直接使用 + 进行字符串的拼接。

3. path 路径模块

3.3 获取路径中的文件名

1. path.basename() 的语法格式

使用 `path.basename()` 方法，可以获取路径中的最后一部分，经常通过这个方法获取路径中的文件名，语法格式如下：

```
1 path.basename(path[, ext])
```

参数解读：

- `path <string>` 必选参数，表示一个路径的字符串
- `ext <string>` 可选参数，表示文件扩展名
- 返回: `<string>` 表示路径中的最后一部分

3. path 路径模块

3.3 获取路径中的文件名

2. path.basename() 的代码示例

使用 path.basename() 方法，可以从一个文件路径中，获取到文件的名称部分：

```
1 const fpath = '/a/b/c/index.html' // 文件的存放路径
2
3 var fullName = path.basename(fpath)
4 console.log(fullName) // 输出 index.html
5
6 var nameWithoutExt = path.basename(fpath, '.html')
7 console.log(nameWithoutExt) // 输出 index
```

3. path 路径模块

3.4 获取路径中的文件扩展名

1. path.extname() 的语法格式

使用 `path.extname()` 方法，可以获取路径中的扩展名部分，语法格式如下：

```
1 path.extname(path)
```

参数解读：

- `path <string>` 必选参数，表示一个路径的字符串
- 返回: `<string>` 返回得到的扩展名字符串

3. path 路径模块

3.4 获取路径中的文件扩展名

2. path.extname() 的代码示例

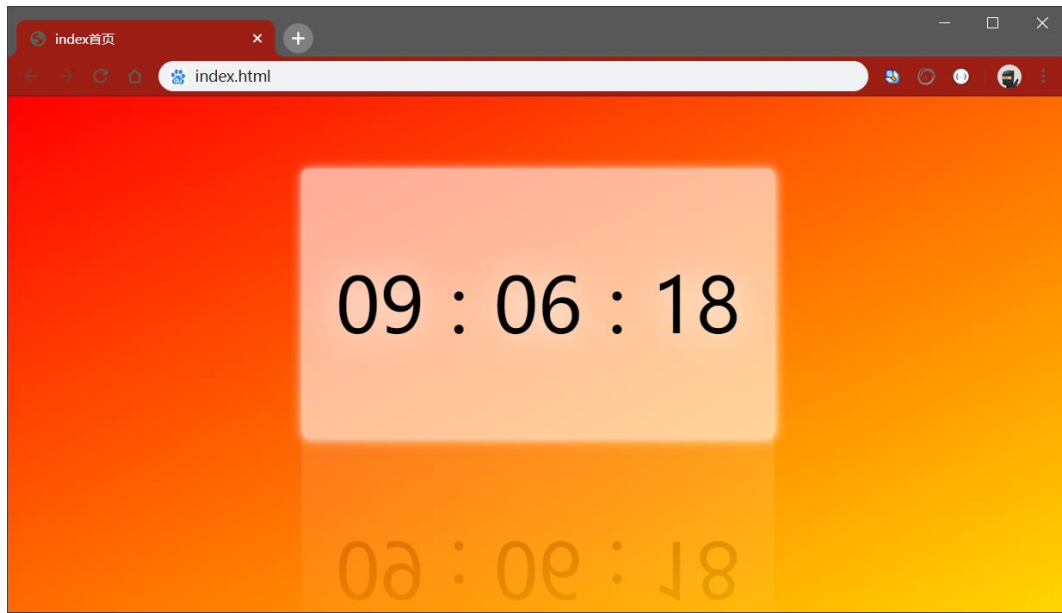
使用 path.extname() 方法，可以获取路径中的扩展名部分：

```
1 const fpath = '/a/b/c/index.html' // 路径字符串
2
3 const fext = path.extname(fpath)
4 console.log(fext) // 输出 .html
```

3. path 路径模块

3.5 综合案例 - 时钟案例

1. 案例要实现的功能



将素材目录下的 index.html 页面，拆分成三个文件，分别是：

- index.css
- index.js
- index.html

并且将拆分出来的 3 个文件，存放到 clock 目录中。

■ 3. path 路径模块

3.5 综合案例 - 时钟案例

2. 案例的实现步骤

- ① 创建两个正则表达式，分别用来匹配 `<style>` 和 `<script>` 标签
- ② 使用 `fs` 模块，读取需要被处理的 HTML 文件
- ③ 自定义 `resolveCSS` 方法，来写入 `index.css` 样式文件
- ④ 自定义 `resolveJS` 方法，来写入 `index.js` 脚本文件
- ⑤ 自定义 `resolveHTML` 方法，来写入 `index.html` 文件



3. path 路径模块

3.5 综合案例 - 时钟案例

3. 步骤1 - 导入需要的模块并创建正则表达式

```
1 // 1.1 导入 fs 文件系统模块
2 const fs = require('fs')
3 // 1.2 导入 path 路径处理模块
4 const path = require('path')
5
6 // 1.3 匹配 <style></style> 标签的正则
7 // 其中 \s 表示空白字符; \S 表示非空白字符; * 表示匹配任意次
8 const regStyle = /<style>[\s\S]*</style>/
9 // 1.4 匹配 <script></script> 标签的正则
10 const regScript = /<script>[\s\S]*</script>/
```



3. path 路径模块

3.5 综合案例 - 时钟案例

3. 步骤2 - 使用 fs 模块读取需要被处理的 html 文件

```
1 // 2.1 读取需要被处理的 HTML 文件
2 fs.readFile(path.join(__dirname, '../素材/index.html'), 'utf8', (err, dataStr) => {
3   // 2.2 读取 HTML 文件失败
4   if (err) return console.log('读取HTML文件失败! ' + err.message)
5
6   // 2.3 读取 HTML 文件成功后, 调用对应的方法, 拆解出 css、js 和 html 文件
7   resolveCSS(dataStr)
8   resolveJS(dataStr)
9   resolveHTML(dataStr)
10 })
```



3. path 路径模块

3.5 综合案例 - 时钟案例

3. 步骤3 – 自定义 resolveCSS 方法

```
1 // 3.1 处理 css 样式
2 function resolveCSS(htmlStr) {
3   // 3.2 使用正则提取页面中的 <style></style> 标签
4   const r1 = regStyle.exec(htmlStr)
5   // 3.3 将提取出来的样式字符串，做进一步的处理
6   const newCSS = r1[0].replace('<style>', '').replace('</style>', '')
7   // 3.4 将提取出来的 css 样式，写入到 index.css 文件中
8   fs.writeFile(path.join(__dirname, './clock/index.css'), newCSS, err => {
9     if (err) return console.log('写入 CSS 样式失败! ' + err.message)
10    console.log('写入 CSS 样式成功! ')
11  })
12 }
```

3. path 路径模块

3.5 综合案例 - 时钟案例

3. 步骤4 – 自定义 resolveJS 方法

```
1 // 4.1 处理 js 脚本
2 function resolveJS(htmlStr) {
3   // 4.2 使用正则提取页面中的 <script></script> 标签
4   const r2 = regScript.exec(htmlStr)
5   // 4.3 将提取出来的脚本字符串，做进一步的处理
6   const newJS = r2[0].replace('<script>', '').replace('</script>', '')
7   // 4.4 将提取出来的 js 脚本，写入到 index.js 文件中
8   fs.writeFile(path.join(__dirname, './clock/index.js'), newJS, err => {
9     if (err) return console.log('写入 JavaScript 脚本失败!' + err.message)
10    console.log('写入 JS 脚本成功! ')
11  })
12 }
```

3. path 路径模块

3.5 综合案例 - 时钟案例

3. 步骤5 – 自定义 resolveHTML 方法

```
1 // 5. 处理 html 文件
2 function resolveHTML(htmlStr) {
3   // 5.1 使用字符串的 replace 方法, 把内嵌的 <style> 和 <script> 标签, 替换为外联的 <link> 和 <script> 标签
4   const newHTML = htmlStr
5     .replace(regStyle, '<link rel="stylesheet" href="./index.css"/>')
6     .replace(regScript, '<script src="./index.js"></script>')
7   // 5.2 将替换完成之后的 html 代码, 写入到 index.html 文件中
8   fs.writeFile(path.join(__dirname, './clock/index.html'), newHTML, err => {
9     if (err) return console.log('写入 HTML 文件失败! ' + err.message)
10    console.log('写入 HTML 页面成功! ')
11  })
12 }
```

■ 3. path 路径模块

3.5 综合案例 - 时钟案例

4. 案例的两个注意点

- ① `fs.writeFile()` 方法只能用来创建文件，不能用来创建路径
- ② 重复调用 `fs.writeFile()` 写入同一个文件，新写入的内容会覆盖之前的旧内容

目录 Contents

- ◆ 初识 Node.js
- ◆ fs 文件系统模块
- ◆ path 路径模块
- ◆ http 模块

4. http 模块

4.1 什么是 http 模块

回顾：什么是客户端、什么是服务器？

在网络节点中，负责消费资源的电脑，叫做客户端；负责对外提供网络资源的电脑，叫做服务器。

http 模块是 Node.js 官方提供的、用来创建 web 服务器的模块。通过 http 模块提供的 `http.createServer()` 方法，就能方便的把一台普通的电脑，变成一台 Web 服务器，从而对外提供 Web 资源服务。

如果要希望使用 http 模块创建 Web 服务器，则需要先导入它：

```
1 const http = require('http')
```


4. http 模块

4.2 进一步理解 http 模块的作用

服务器和普通电脑的**区别**在于，服务器上安装了 **web 服务器软件**，例如：IIS、**Apache** 等。通过安装这些服务器软件，就能把一台普通的电脑变成一台 web 服务器。

在 Node.js 中，我们**不需要使用** IIS、Apache 等这些**第三方 web 服务器软件**。因为我们可以基于 Node.js 提供的 http 模块，**通过几行简单的代码，就能轻松的手写一个服务器软件**，从而对外提供 web 服务。

4.3 服务器相关的概念

1. IP 地址

IP 地址就是互联网上**每台计算机的唯一地址**，因此 IP 地址具有唯一性。如果把“个人电脑”比作“一台电话”，那么“IP地址”就相当于“电话号码”，只有在知道对方 IP 地址的前提下，才能与对应的电脑之间进行数据通信。

IP 地址的格式：通常用“**点分十进制**”表示成 (a.b.c.d) 的形式，其中，a,b,c,d 都是 0~255 之间的十进制整数。例如：用点分十进表示的 IP 地址 (192.168.1.1)

注意：

- ① **互联网中每台 Web 服务器，都有自己的 IP 地址**，例如：大家可以在 Windows 的终端中运行 `ping www.baidu.com` 命令，即可查看到百度服务器的 IP 地址。
- ② 在开发期间，自己的电脑既是一台服务器，也是一个客户端，为了方便测试，可以在自己的浏览器中输入 127.0.0.1 这个 IP 地址，就能把自己的电脑当做一台服务器进行访问了。

4.3 服务器相关的概念

2. 域名和域名服务器

尽管 IP 地址能够唯一地标记网络上的计算机，但IP地址是一长串数字，**不直观**，而且**不便于记忆**，于是人们又发明了另一套**字符型的地址方案**，即所谓的**域名 (Domain Name) 地址**。

IP地址和域名是一一对应的关系，这份对应关系存放在一种叫做**域名服务器**(DNS, Domain name server)的电脑中。使用者只需通过好记的域名访问对应的服务器即可，对应的转换工作由域名服务器实现。因此，**域名服务器就是提供 IP 地址和域名之间的转换服务的服务器**。

注意：

- ① 单纯使用 IP 地址，互联网中的电脑也能够正常工作。但是有了域名的加持，能让互联网的世界变得更加方便。
- ② 在开发测试期间，**127.0.0.1** 对应的域名是 **localhost**，它们都代表我们自己的这台电脑，在使用效果上没有任何区别。

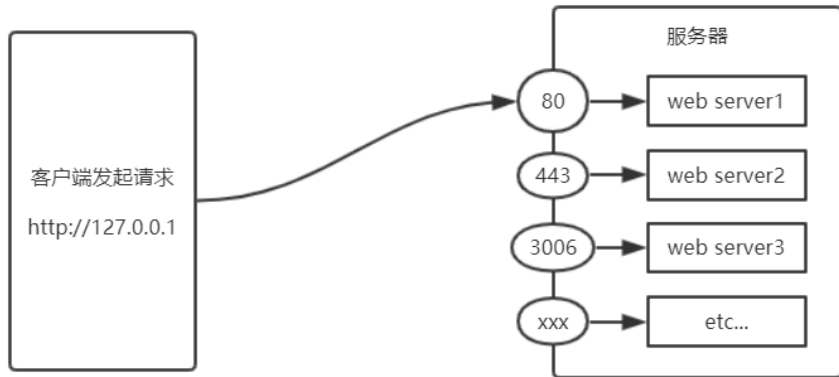
4. http 模块

4.3 服务器相关的概念

3. 端口号

计算机中的端口号，就好像是现实生活中的门牌号一样。通过门牌号，外卖小哥可以在整栋大楼众多的房间中，准确把外卖送到你的手中。

同样的道理，在一台电脑中，可以运行成百上千个 web 服务。每个 web 服务都对应一个唯一的端口号。客户端发送过来的网络请求，通过端口号，可以被准确地交给对应的 web 服务进行处理。



注意：

- ① 每个端口号不能同时被多个 web 服务占用。
- ② 在实际应用中，URL 中的 80 端口可以被省略。

4. http 模块

4.4 创建最基本的 web 服务器

1. 创建 web 服务器的基本步骤

- ① 导入 http 模块
- ② 创建 web 服务器实例
- ③ 为服务器实例绑定 **request** 事件，[监听客户端的请求](#)
- ④ 启动服务器

4. http 模块

4.4 创建最基本的 web 服务器

2. 步骤1 - 导入 http 模块

如果希望在自己的电脑上创建一个 web 服务器，从而对外提供 web 服务，则需要导入 http 模块：

```
1 const http = require('http')
```

4. http 模块

4.4 创建最基本的 web 服务器

2. 步骤2 - 创建 web 服务器实例

调用 `http.createServer()` 方法，即可快速创建一个 web 服务器实例：

```
1 const server = http.createServer()
```

4. http 模块

4.4 创建最基本的 web 服务器

2. 步骤3 - 为服务器实例绑定 request 事件

为服务器实例绑定 request 事件，即可监听客户端发送过来的网络请求：

```
1 // 使用服务器实例的 .on() 方法，为服务器绑定一个 request 事件
2 server.on('request', (req, res) => {
3   // 只要有客户端来请求我们自己的服务器，就会触发 request 事件，从而调用这个事件处理函数
4   console.log('Someone visit our web server.')
5 })
```


4. http 模块

4.4 创建最基本的 web 服务器

2. 步骤4 - 启动服务器

调用服务器实例的 `.listen()` 方法，即可启动当前的 web 服务器实例：

```
1 // 调用 server.listen(端口号, cb回调) 方法，即可启动 web 服务器
2 server.listen(80, () => {
3   console.log('http server running at http://127.0.0.1')
4 })
```

4. http 模块

4.4 创建最基本的 web 服务器

3. req 请求对象

只要服务器接收到了客户端的请求，就会调用通过 `server.on()` 为服务器绑定的 `request` 事件处理函数。

如果想在事件处理函数中，访问与客户端相关的**数据或属性**，可以使用如下的方式：

```
1 server.on('request', (req) => {  
2   // req 是请求对象，它包含了与客户端相关的数据和属性，例如：  
3   // req.url 是客户端请求的 URL 地址  
4   // req.method 是客户端的 method 请求类型  
5   const str = `Your request url is ${req.url}, and request method is ${req.method}`  
6   console.log(str)  
7 })
```

4. http 模块

4.4 创建最基本的 web 服务器

4. res 响应对象

在服务器的 request 事件处理函数中，如果想访问与服务器相关的数据或属性，可以使用如下的方式：

```
1 server.on('request', (req, res) => {
2   // res 是响应对象，它包含了与服务器相关的数据和属性，例如：
3   // 要发送到客户端的字符串
4   const str = `Your request url is ${req.url}, and request method is ${req.method}`
5   // res.end() 方法的作用：
6   // 向客户端发送指定的内容，并结束这次请求的处理过程
7   res.end(str)
8 })
```

4. http 模块

4.4 创建最基本的 web 服务器

5. 解决中文乱码问题

当调用 `res.end()` 方法，向客户端发送中文内容的时候，会出现乱码问题，此时，需要手动设置内容的编码格式：

```
1 server.on('request', (req, res) => {
2   // 发送的内容包含中文
3   const str = `您请求的 url 地址是 ${req.url}, 请求的 method 类型是 ${req.method}`
4   // 为了防止中文显示乱码的问题，需要设置响应头 Content-Type 的值为 text/html; charset=utf-8
5   res.setHeader('Content-Type', 'text/html; charset=utf-8')
6   // 把包含中文的内容，响应给客户端
7   res.end(str)
8 })
```

4. http 模块

4.5 根据不同的 url 响应不同的 html 内容

1. 核心实现步骤

- ① 获取请求的 url 地址
- ② 设置默认响应内容为 404 Not found
- ③ 判断用户请求的是否为 / 或 /index.html 首页
- ④ 判断用户请求的是否为 /about.html 关于页面
- ⑤ 设置 Content-Type 响应头，防止中文乱码
- ⑥ 使用 res.end() 把内容响应给客户端

4. http 模块

4.5 根据不同的 url 响应不同的 html 内容

2. 动态响应内容

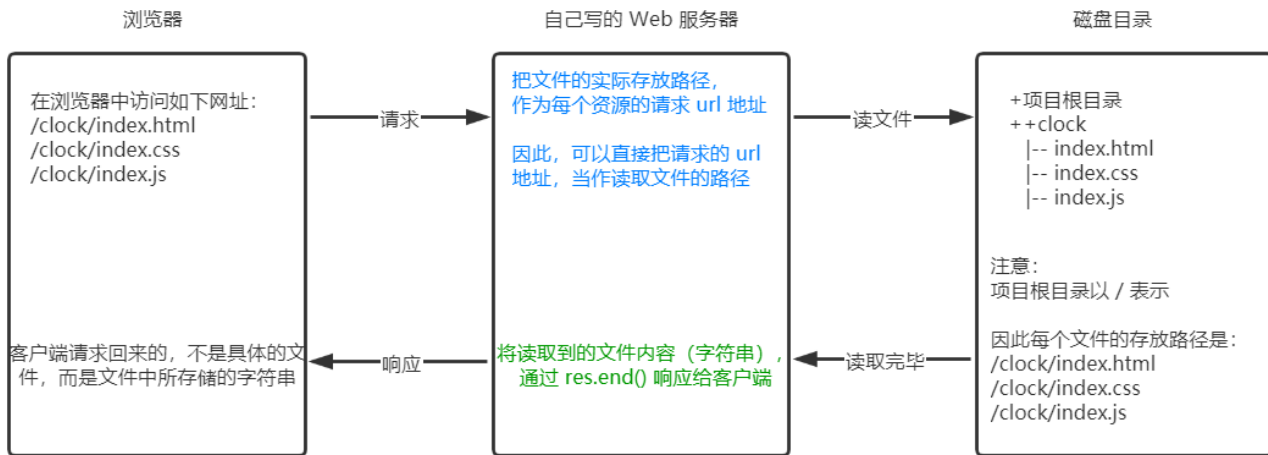
```
1 server.on('request', function(req, res) {
2   const url = req.url // 1. 获取请求的 url 地址
3   let content = '<h1>404 Not found!</h1>' // 2. 设置默认的内容为 404 Not found
4   if (url === '/' || url === '/index.html') {
5     content = '<h1>首页</h1>' // 3. 用户请求的是首页
6   } else if (url === '/about.html') {
7     content = '<h1>关于页面</h1>' // 4. 用户请求的是关于页面
8   }
9   res.setHeader('Content-Type', 'text/html; charset=utf-8') // 5. 设置 Content-Type 响应头, 防止中文乱码
10  res.end(content) // 6. 把内容发送给客户端
11 })
```

4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

1. 核心思路

把文件的实际存放路径，作为每个资源的请求 url 地址。



服务器充当的角色
就是一个字符串的搬运工

■ 4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

2. 实现步骤

- ① 导入需要的模块
- ② 创建基本的 web 服务器
- ③ 将资源的请求 url 地址映射为文件的存放路径
- ④ 读取文件内容并响应给客户端
- ⑤ 优化资源的请求路径

4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

3. 步骤1 - 导入需要的模块

```
1 // 1.1 导入 http 模块
2 const http = require('http')
3 // 1.2 导入 fs 文件系统模块
4 const fs = require('fs')
5 // 1.3 导入 path 路径处理模块
6 const path = require('path')
```

4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

3. 步骤2 - 创建基本的 web 服务器

```
1 // 2.1 创建 web 服务器
2 const server = http.createServer()
3
4 // 2.2 监听 web 服务器的 request 事件
5 server.on('request', function(req, res) { })
6
7 // 2.3 启动 web 服务器
8 server.listen(80, function() {
9   console.log('server listen at http://127.0.0.1')
10 })
```

4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

3. 步骤3 - 将资源的请求 url 地址映射为文件的存放路径

```
1 // 3.1 获取到客户端请求的 url 地址
2 const url = req.url
3 // 3.2 把 请求的 url 地址, 映射为本地文件的存放路径
4 const fpath = path.join(__dirname, url)
```

4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

3. 步骤4 - 读取文件的内容并响应给客户端

```
1 // 4.1 根据“映射”过来的文件路径读取文件
2 fs.readFile(fpath, 'utf8', (err, dataStr) => {
3   // 4.2 读取文件失败后，向客户端响应固定的“错误消息”
4   if(err) return res.end('404 Not found.')
5   // 4.3 读取文件成功后，将“读取成功的内容”响应给客户端
6   res.end(dataStr)
7 })
```

4. http 模块

4.6 案例 - 实现 clock 时钟的 web 服务器

3. 步骤5 – 优化资源的请求路径

```
1  // *** 将 3.2 的实现方式, 改为如下代码↓↓↓ ***
2  // 5.1 预定义空白的文件存放路径
3  let fpath = ''
4  if (url === '/') {
5      // 5.2 如果请求的路径是否为 /, 则手动指定文件的存放路径
6      fpath = path.join(__dirname, './clock/index.html')
7  } else {
8      // 5.3 如果请求的路径不为 /, 则动态拼接文件的存放路径
9      fpath = path.join(__dirname, './clock', url)
10 }
```



传智播客旗下高端IT教育品牌