Node 简介

客户端的JavaScript是怎样的

- 什么是 JavaScript?
  +是一个脚本语言
  +运行在浏览器（浏览器的js解析内核 v8）
  +实现用户的交互 （interactive）
  - 变量 赋值 循环 逻辑 判断 分支 对象 函数。。。。
  - dom 操作
  - bom 操作
  - ajax

- JavaScript 的运行环境?
  +浏览器内核解析内核 es6

- 浏览器中的 JavaScript 可以做什么?

- 浏览器中的 JavaScript 不可以做什么？（不安全）
  +访问数据库
  +不能对文件进行操作
  +对os 进行操作
  +原因 是不安全 和浏览器运行机制有关

- 在开发人员能力相同的情况下编程语言的能力取决于什么?

  +cordova hbuilder 平台 platform
  +java java虚拟机 （运行平台）
  +php php虚拟机
  +c# .net framework mono
  +js 解析内核 chrome v8

- JavaScript 只可以运行在浏览器中吗?
  +不是

为什么是JavaScript

- node js 不是因为js 产生的
- node 选择了js
- Ryan dahl

- 2009 2 月份 node有想法
- 2009 5 月份 githup 开源
- 2009 11月份 jsconf 讲解推广node
- 2010年底 被xxx公司收购
- 2018 发布有重大bug
- npm
- githup 世界上最大的同性交友网站 码云

what is node ?

- Node.js 是一个基于Chrome V8 引擎的JavaScript运行环境
- Node.js使用了一个事件驱动、非阻塞式I/O的模型,使其轻量又高效
- Node.js的包管理工具npm,是全球最大的开源库生态系统
- 官网 http://nodejs.cn/
- npm 插件官网：https://www.npmjs.com/

环境配置

Node的安装

- 安装包安装
  - 官网下载对应的安装包
  - 一路next

- nvm安装(有一个类似的工具：n)

  - Node Version Manager（Node版本管理工具）
  - 由于以后的开发工作可能会在多个Node版本中测试，而且Node的版本也比较多，所以需要这么款工具来管理
    +

相关版本

- node版本常识

  - 偶数版本为稳定版 （0.6.x ， 0.8.x ， 0.10.x）
  - 奇数版本为非稳定版（0.7.x ， 0.9.x ， 0.11.x）
  - LTS（Long Term Support）
  - LTS和Current区别
- 操作方式：
  - 重新下载最新的安装包；

    - 覆盖安装即可；
- 问题：
    - 以前版本安装的很多全局的工具包需要重新安装
    - 无法回滚到之前的版本
    - 无法在多个版本之间切换（很多时候我们要使用特定版本）

Windows下常用的命令行操作

- 切换当前目录（change directory）：cd
- 创建目录（make directory）：mkdir
- 查看当前目录列表（directory）：dir
    - 别名：ls（list）
- 清空当前控制台：cls
    - 别名：clear
- 删除文件：del
    - 别名：rm

> *注意：所有别名必须在新版本的 PowerShell （linux系统）中使用*

常见问题

- Python环境丢失
    - Node中有些第三方的包是以C/C++源码的方式发布的，需要安装后编译,确保全局环境中可以使用python命令,python 版本推荐2.7.0

- 环境变量丢失

    - 部分电脑安装完毕之后没有环境变量需要手动配置
    - Windows中环境变量分为系统变量和用户变量
    - 环境变量的变量名是不区分大小写的
    - PATH 变量：只要添加到 PATH 变量中的路径，都可以在任何目录下
    - 目的可以在任何地方调起node命令

    >

模块,包 commonjs

commonjs规范

前端模块化：AMD,CMD,Commonjs

Node 应用由模块组成，采用 CommonJS 模块规范。

定义module

每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见。

暴露接口

CommonJS规范规定，每个模块内部，module变量代表当前模块。这个变量是一个对象，它的exports属性（即module.exports）是对外的接口。加载某个模块，其实是加载该模块的module.exports属性。

```
var x = 5;
var addX = function (value) {
return value + x;
};
module.exports.x = x;
module.exports.addX = addX;
```

引用

require方法用于加载模块。

```
var example = require('./example.js');
console.log(example.x); // 5
console.log(example.addX(1)); // 6
```

模块的分类

- 内置模块

```
const process = require('process')
const path = require('path')
console.log(process.version)
console.log(path.resolve('../'))
```

- 第三方模块

```
const request=require("request");
console.log(request)
request.get('http://api.douban.com/v2/movie/in_theaters', (err, response, body) =>
{
if (!err) {
// console.log(body);
console.log(JSON.parse(body))
} else {
```

```
    console.log(err);
  }
})
```

- 自定义模块

npm 使用入门

官网:https://www.npmjs.com/

安装：无需安装

查看当前版本：

```
$ npm -v
```

更新：

```
$ npm install npm@latest -g
```

初始化工程

```
$ npm init

$ npm init --yes 默认配置
```

安装包

使用npm install会读取package.json文件来安装模块。安装的模块分为两类dependencies和devDependencies，分别对应生产环境需要的安装包和开发环境需要的安装包。

```
$ npm install

$ npm install <package_name>

$ npm install <package_name> --save

$ npm install <package_name> --save-dev
```

更新模块

```
$ npm update
```

卸载模块

```
$ npm uninstall <package_name>

$ npm uninstall --save lodash
```

配置npm源

- 临时使用，安装包的时候通过—registry参数即可

```
$ npm install express --registry https://registry.npm.taobao.org
```

- 全局使用

```
$ npm config set registry https://registry.npm.taobao.org
// 配置后可通过下面方式来验证是否成功
npm config get registry
// 或
npm info express
```

- cnpm 使用

```
// 安装cnpm
npm install -g cnpm --registry=https://registry.npm.taobao.org

// 使用cnpm安装包
cnpm install express
```

>

常用的内置模块

node 常用内置api

(1) URL 网址解析
解析URL相关网址信息
url.parse(urlString[, parseQueryString[, slashesDenoteHost]])
url.format(urlObject)
url.resolve(from, to)
(2) QueryString 参数处理
querystring.escape(str)
querystring.unescape(str)
querystring.parse(str[, sep[, eq[, options]]])
querystring.stringify(obj[, sep[, eq[, options]]])
(3) HTTP 模块概要
http.createServer([options][, requestListener])
http.get(options[, callback])
简易的爬虫
代理跨域处理
(4) 事件 events 模块
(5) 文件fs模块

(5) 文件fs模块
打印目录树
(6) Stream 流模块
歌词播放
音乐下载
(8) request 方法

2、Node.js 基础应用
1、应用 HTTP 模块编写一个小爬虫工具
(1) 利用爬虫获取“拉勾网”首页列表数据
(2) 通过 npm 安装 cheerio 模块获得数据
2、后端表单的提交
要求:
(1) 应用 request post 模拟提交表单

文件读取

Node中文件读取的方式主要有：

> *fs.readFile(file[, options], callback(error, data))*

```
fs.readFile('c:\\demo\1.txt', 'utf8', (err, data) => {
 if (err) throw err;
 console.log(data);
});
```

> *fs.readFileSync(file[, options])*

```
try {
 const data = fs.readFileSync('c:\\demo\1.txt', 'utf8');
 console.log(data);
} catch(e) {
 // 文件不存在，或者权限错误
 throw e;
}
```

> *fs.createReadStream(path[, options])*

```
const stream = fs.createReadStream('c:\\demo\1.txt');
let data = ''
stream.on('data', (trunk) => {
 data += trunk;
});
stream.on('end', () => {
 console.log(data);
```

```
    console.log(data);
});
```

由于Windows平台下默认文件编码是GBK，在Node中不支持，可以通过iconv-lite解决

Readline模块逐行读取文本内容

```
const readline = require('readline');
const fs = require('fs');

const rl = readline.createInterface({
  input: fs.createReadStream('sample.txt')
});

rl.on('line', (line) => {
  console.log('Line from file:', line);
});
```

文件写入

Node中文件写入的方式主要有：

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

```
fs.writeFile('c:\\demo\a.txt', new Date(), (error) => {
  console.log(error);
});
```

fs.writeFileSync(file, data[, options])

```
try {
  fs.writeFileSync('c:\\demo\a.txt', new Date());
} catch (error) {
  // 文件夹不存在，或者权限错误
  console.log(error);
}
```

fs.createWriteStream(path[,option])

```
var streamWriter = fs.createWriteStream('c:\\demo\a.txt');
setInterval(() => {
  streamWriter.write(`${new Date}\n`, (error) => {
  console.log(error);
  });
}, 1000);
```

node中的异步操作

- 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()
- 区别：
  - 同步调用会阻塞代码的执行，异步则不会
  - 异步调用会将读取任务下达到任务队列，直到任务执行完成才会回调
  - 异常处理方面，同步必须使用 try catch 方式，异步可以通过回调函数的第一个参数

```
1.  console.time('sync');
2.  try {
3.   var data = fs.readFileSync(path.join('C:\\Users\\iceStone\\Downloads', 'H.mp4'));
4.   // console.log(data);
5.  } catch (error) {
6.   throw error;
7.  }
8.  console.timeEnd('sync');
9.
10.  console.time('async');
11.  fs.readFile(path.join('C:\\Users\\iceStone\\Downloads', 'H.mp4'), (error, data) => {
12.   if (error) throw error;
13.   // console.log(data);
14.  });
15.  console.timeEnd('async');
```

promise 对象的使用

参考资料：JavaScript Promise迷你书

- what is Promise *
  Promise是抽象异步处理对象以及对其进行各种操作的组件。Promise并不是从JavaScript中发祥的概念。
  Promise最初被提出是在 E语言中， 它是基于并列/并行处理设计的一种编程语言。
  现在JavaScript也拥有了这种特性，这就是JavaScript Promise

使用了回调函数的异步处理

```
1.
2.  ----
3.  getAsync("fileA.txt", function(error, result){
4.   if(error){// 取得失败时的处理
5.    throw error;
```

```
  }
  // 取得成功时的处理
});
----
<1> 传给回调函数的参数为(error对象， 执行结果)错误优先处理
```

使用了回调函数的异步处理

```
----
var promise = getAsyncPromise("fileA.txt");
promise.then(function(result){
  // 获取文件内容成功时的处理
}).catch(function(error){
  // 获取文件内容失败时的处理
});
----
<1> 返回promise对象
```

- 创建Promise对象 *

```
var promise = new Promise(function(resolve, reject) {
  // 异步处理
  // 处理结束后、调用resolve 或 reject
  resolve('成功处理')
  reject('错误处理')
});
```

- 使用实例 *

1. 创建一个priomise 对象并返回 `new Promise(fn)`
2. 在fn 中指定异步等处理
   - 处理结果正常的话，调用 `resolve(处理结果值)`
   - 处理结果错误的话，调用 `reject(Error对象)`

```
function asyncFunction() {

  return new Promise(function (resolve, reject) {
  setTimeout(function () {
  resolve('Async Hello world');
  }, 16);
  });
}

asyncFunction().then(function (value) {
  console.log(value); // => 'Async Hello world'
```

```
12.  }).catch(function (error) {
13.    console.log(error);
14.  });
```

- Promise的状态

用new Promise 实例化的promise对象有以下三个状态。

- **“has-resolution” - Fulfilled** resolve(成功)时。
- **“has-rejection” - Rejected** reject(失败)时
- **“unresolved” - Pending** 既不是resolve也不是reject的状态。也就是promise对象刚被创建后的初始化状态等

promise对象的状态，从Pending转换为Fulfilled或Rejected之后， 这个promise对象的状态就不会再发生任何变化。

也就是说，Promise与Event等不同，在.then 后执行的函数可以肯定地说只会被调用一次。

另外，Fulfilled和Rejected这两个中的任一状态都可以表示为Settled(不变的)。

Settled
resolve(成功) 或 reject(失败)。

从Pending和Settled的对称关系来看，Promise状态的种类/迁移是非常简单易懂的。

当promise的对象状态发生变化时，用.then 来定义只会被调用一次的函数。

路径模块

在文件操作的过程中，都必须使用物理路径（绝对路径），path模块提供了一系列与路径相关的 API

```
1.  console.log('join用于拼接多个路径部分，并转化为正常格式');
2.  const temp = path.join(__dirname, '..', 'lyrics', './友谊之光.lrc');
3.  console.log(temp);
4.
5.  console.log('获取路径中的文件名');
6.  console.log(path.basename(temp));
7.
8.  console.log('获取路径中的文件名并排除扩展名');
9.  console.log(path.basename(temp, '.lrc'));
10.
11.  console.log('====================================');
12.
```

```
13. console.log('获取不同操作系统的路径分隔符');
14. console.log(process.platform + '的分隔符为 ' + path.delimiter);
15.
16. console.log('一般用于分割环境变量');
17. console.log(process.env.PATH.split(path.delimiter));
18.
19. console.log('====================================');
20.
21. console.log('获取一个路径中的目录部分');
22. console.log(path.dirname(temp));
23.
24. console.log('====================================');
25.
26. console.log('获取一个路径中最后的扩展名');
27. console.log(path.extname(temp));
28.
29. console.log('====================================');
30.
31. console.log('将一个路径解析成一个对象的形式');
32. const pathObject = path.parse(temp);
33. console.log(pathObject);
34.
35. console.log('====================================');
36.
37. console.log('将一个路径对象再转换为一个字符串的形式');
38. // pathObject.name = '我终于失去了你';
39. pathObject.base = '我终于失去了你.lrc';
40. console.log(pathObject);
41.
42. console.log(path.format(pathObject));
43.
44. console.log('====================================');
45.
46. console.log('获取一个路径是不是绝对路径');
47. console.log(path.isAbsolute(temp));
48. console.log(path.isAbsolute('../lyrics/爱的代价.lrc'));
49.
50. console.log('====================================');
51.
52. console.log('将一个路径转换为当前系统默认的标准格式，并解析其中的./和../');
```

```
53. console.log(path.normalize(' c:/develop/demo\\hello/../world/./a.txt '));
54. 
55. console.log('====================================');
56. 
57. console.log('获取第二个路径相对第一个路径的相对路径');
58. console.log(path.relative(__dirname, temp));
59. 
60. console.log('====================================');
61. 
62. console.log('以类似命令行cd命令的方式拼接路径');
63. console.log(path.resolve(temp, 'c:/', './develop', '../application'));
64. 
65. console.log('====================================');
66. 
67. console.log('获取不同平台中路径的分隔符（默认）');
68. console.log(path.sep);
69. 
70. console.log('====================================');
71. 
72. console.log('允许在任意平台下以WIN32的方法调用PATH对象');
73. // console.log(path.win32);
74. console.log(path === path.win32);
75. 
76. console.log('====================================');
77. 
78. console.log('允许在任意平台下以POSIX的方法调用PATH对象');
79. console.log(path === path.posix);
```