高级node第四单元（6.1）

2020年6月1日 9:00

模拟anywhere实现静态服务器

1. 向全局注册一个命令 pro -s
2. 在引用执行的js文件中启动服务器
3. 自动打开浏览器
   使用子进程的exec方法调用 start 地址的方式

```
D:\software\phpStudy\WWW\1909A-高
λ chrome https://www.baidu.com

D:\software\phpStudy\WWW\1909A-高
λ chrome http://localhost:3000

D:\software\phpStudy\WWW\1909A-高
λ start https://www.baidu.com
```

```
const express = require( express );
const childProcess = require('child_process');

// 目动打开浏览器的动作
childProcess.exec('start http://localhost:3000')
```

服务端渲染

1. 在服务端直接根据数据生成对应的页面文件
2. ejs就是一个用来做服务端渲染的技术
   a. 编写视图模板

```
<body>
  <h1>总共有<%= fileLists.length %>个链接</h1>
  <ul>
    <% fileLists.forEach(item => { %>
    <li>
      <a href="<%= item.href %>"><%= item.title %></a>
    </li>
    <% }) %>
  </ul>
</body>
```

   b. 调用ejs进行渲染，并提供渲染数据

```
//
let dirListHtml = fs.readFileSync(path.join(__dirname, './view/dirList.html'), 'utf-8');
res.send(ejs.render(dirListHtml, { fileLists }));
```

作业：编写一个类似于cli-dict的命令行工具

url地址中的合法字符串和不合法字符串

1. url地址不会识别所有的字符串，比如 中文，特殊的表单符号是不可以识别的
2. url地址中出现了不合法的字符串，就会被就行url编码

```
encodeURIComponent('我们')
"%E6%88%91%E4%BB%AC"
```

3. 在使用地址栏的数据的时候，需要解码

```
decodeURIComponent('%E6%88%91%E4%BB%AC')
"我们"
```

4. 浏览器一般会在内部对url中不合法的字符串进行url编码

os模块

1. os模块的作用

os 模块提供了与操作系统相关的实用方法和属性。 使用方法如下：

```
const os = require('os');
```

2. api查阅文档

Fs.createReadStream和fs.createWriteStream

1. createReadStream创建的可读流 从文件系统 => 内存
2. createWriteStream创建的可写流 内存 => 文件系统

createReadStream流的相关用法

1. 具体用法如下-可读流的流动

```
const fs = require('fs')
// 创建一个可读流，从abc.txt文件 => 内存
const rs = fs.createReadStream('./abc.txt', {
  highWaterMark: 2048
})
function isOpenFile() {
  return rs.pending ? '文件还没有打开' : '文件已经打卡了';
}
console.log(isOpenFile());
// 监听该文件是否打开
rs.on('open', () => {
  console.log(`abc.txt已经打开了`);
  console.log(isOpenFile())
})
rs.on('ready', () => {
  console.log('ready-可以读取数据了');
})
// 监听data事件，然后流就会不断的触发该事件
rs.on('data', chunk => {
  console.count('rs-data');
  console.log(`当前已经读取了${rs.bytesRead}`)
})
rs.on('end', () => {
  console.log('abc.txt文件已经关闭')
  console.log(isOpenFile())
})
```

2. 流是可以在中途中断和恢复的
   pause () 暂停流
   resume() 恢复流

```
//  if (rs.bytesRead === 2048) {
//    rs.pause();
//  }

// rs.resume();
```

createWriteStream流的相关用法

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

let ws = fs.createWriteStream('./abc.txt');
ws.write('a', 'utf-8')
ws.write('a', 'utf-8')
ws.end();
ws.on('open', () => {
  console.log('文件已经打开')
})
ws.on('ready', () => {
  console.log('ready-开始把数据流入abc.txt');
})

ws.on('close', () => {
  console.log('close-文件关闭了');
})
```

以流的方式实现拷贝

1. 直接使用流的api

```
const fs = require('fs');
// 创建可读流
const rs = fs.createReadStream('../package.json', {
  highWaterMark: 1024
})
// 创建可写流
const ws = fs.createWriteStream('./abc.json');
// 数据从可读流中流入到可写流
rs.on('data', chunk => {
  ws.write(chunk)
})
// 可读流数据读完
rs.on('end', () => {
  console.log('end')
  ws.end();
})
```

2. 流的管道方法-pipe

```
const fs = require('fs');
// 创建可读流
const rs = fs.createReadStream('../package.json', {
  highWaterMark: 1024
})
// 创建可写流
const ws = fs.createWriteStream('./abc.json');

rs.pipe(ws);
```