高级node第十四单元（6.17）

2020年6月17日 9:51

总结与复习

Child_process模块

项目功能：实现自定义头像

Child_process模块

可以在主进程下衍生一个子进程去执行任务，原则上子进程和主进程是一样的，没有什么区别

有如下方法：

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

const {
  spawn, // 提供一个环境可以执行对应的命令
  // 以下的三个方法底层都是依赖 spawn方法
  exec, // 提供一个shell环境，执行命令
  execFile, // 与exec基本相似，直接运行可执行文件
  fork // 专门用来运行node程序的
} = childProcess;
```

spawn的用法

Spanwn(commander<string>,[argv<array<string>>, options])

Return <ChildProcess>

```
// 使用spawn运行 ls -a命令

const ls = spawn('ls', ['-a'])
// console.log(ls);
ls.stdout.on('data', chunk => {
  // console.log(chunk.toString());
})

// 使用spawn运行 dir 命令

const dir = spawn('dir');

dir.stdout.on('data', chunk => {
  // console.log(chunk.toString())
})
```

exec和execFile的用法

child_process.exec(command[, options][, callback])

中英对照 提交修改

- `command` <string> 要运行的命令，参数使用空格分隔。
- `callback` <Function> 当进程终止时调用并传入输出。
  - `error` <Error>
  - `stdout` <string> | <Buffer>
  - `stderr` <string> | <Buffer>

`child_process.execFile()` 函数类似于 `child_process.exec()`，但默认情况下不会衍生 shell，而且可以执行文件 `file` 会被直接衍生作为新的进程，使其比 `child_process.exec()` 稍微更高效。

```
// 使用exec执行a.js
const execIndex = exec('node a.js', (err, stdout, stderr) => {
  console.log(stdout);
})
```

Fork的使用

child_process.fork(modulePath[, args][, options])

```
const a = fork('a.js');
```

进程通信

send发型消息

On('message', msg => {})接收消息

进程阻塞的例子

1.子进程的用法.js 2.fork.js 3.server-multipart.js a.js

```
const http = require('http');
const { fork } = require('child_process')

http.createServer((req, res) => {
  if (req.url === '/computed') {
    const a = fork('./a.js');
    a.on('message', msg => {
      console.log(msg);
      res.end('computed');
    })

  } else if (req.url === '/normal') {
    res.end('normal');
  }
}).listen(3000);
```

1.子进程的用法.js 2.fork.js 3.server-multipart.js a.js

```
for (let i = 0; i < 30000; i++) {
  console.log(i);
}

process.send('ok');
```