Nodejs 学习

一、认识 nodeJs

1、Node 给 JavaScript 带来的意义

(1) 在 Node 中，JavaScript 可以随心所欲地访问本地文件，可以搭建 WebSocket 服务器端，可以连接数据库，可以如 Web Workers 一样玩转多进程

(2) Node 打破了过去 JavaScript 只能在浏览器中运行的局面

(3) 前后端变成环境统一，大大降低前后端转换所需要的上下文大家

2、Node 特点

(1) 异步 I/O

每个调用之间无需等待之前的 I/O 调用结束

(2) 事件与回调函数

将前端浏览器成熟事件引入后端，配合异步 I/O，将事件点暴露给业务

回调函数也是最好的接受异步调用返回数据的方式，代码编写顺序与执行顺序并无关系

(3) 单线程

单线程弱点：

无法利用多核 CPU

错误会引起整个应用退出

大量计算占用 CPU 导致无法继续调用异步 I/O

解决办法：通过子进程，将计算分发给各个紫禁城，在通过进程之间事件消息传递结果。

(4) 跨平台

二、模块机制

1. 模块引入和定义 require() 和 exports.属性

2. Module.exports 和 exports 区别

```
// 1.
exports.add = function() {
    var sum = 0;
    i = 0;
    args = arguments,
    l = args.length;
    while (l-- > 0) {
        sum += args[i++];
    }
    return sum;
}
// 2.
var math = require('./03.cmd_1');
console.log(math.add(1, 3, 7)); //11 使用 math.add
```

```
// module.exports = function () {
//     console.log('空函数');
// }

var math = require('./03.cmd_1');
math();  // 空函数
// 3.
exports = function () {
        console.log('空函数');
    }
var math = require('./03.cmd_1');
math();  // 报错
// 4.
// module.exports.ex = function () {
//     console.log('空函数');
// }
var math = require('./03.cmd_1');
Math.ex();  // 空函数
// 综上可以看出 exports 只是一个 对 module.exports 的引
用，  exports --> module.exports，
```

3．Node 对 JavaScript 内容进行了头尾包装，

```
(function (exports, require, module, __filename, __dirname) {
    ......
    };
   });
```

4．全局安装并不是将一个模块包安装为一个全局包，二十在包描述文件中 bin 字段配置，将实际脚本链接到与 Node 可执行文件相同的路径。

三、异步 I/O

1．为什么要用异步 I/O

（1）随着页面复杂性增加，同步的时间消耗总和 M+N+.....

（2）异步的时间消耗总和 max(M,N,...)

2．资源分配， 利用单线程，原理多线程死锁，状态同步，利用异步 I/O，然单线程远离阻塞，以根号使用 CPU

3．阻塞 I/O 和非阻塞 I/O

（1）阻塞 I/O 是要等待系统内核层面完成所有操作后，调用才结束，浪费等待时间

（2）非阻塞 I/O 在调用之后会不带数据直接返回，获取数据还需要通过文件描述符再次获取，为了获取完整的数据，应用程序会重复调用 I/O 来确认操作是否完成（I 轮询）

4．理想的非阻塞异步 I/O

通过让部分线程进行阻塞 I/O 或者非阻塞 I/O 加轮询技术完成数据获取，让一个线程进行计算处理，通过线程之间的通信将 I/O 得到 的数据进行传递。分配任务处理结果的线程

是大关机，I/O 线程池里的各个 I/O 都是老二，老二和管家之间互不依赖。

四、node 基本模块

1. fs

```
'use strict';

var fs = require('fs');
// 参数:  文件路径,  编码格式,  处理函数
// fs.readFile('01helloWorld.js', 'utf-8', function(err, data) {
//     if(err) {
//         console.log(err);
//     }else {
//         console.log(data);
//     }
// })
```

```
// 获取二进制文件
// fs.readFile('01.docx', (err, data) => {
//     if(err) {
//         console.log(err);
//     }else {
//         console.log(data);
//         console.log(data.length + 'bytes');
//         // 二进制文件返回的 data 是一个 Buffer 对象，可以转换成 String 对象
//         let text = data.toString('utf-8');
//         console.log(text);
//         // 可以将 String 转换成 Buffer 对象
//         let buf = Buffer.from(text, 'utf-8');
//         console.log(buf);
//     }
// })
```

```
// 同步读文件,  同步读取的函数和异步相比，多了一个 Sync 后缀，并且不接受回调函数，函数直接返回结果
// var data = fs.readFileSync('01.helloWorld.js', 'utf-8');
// console.log(data);
// 发生错误需要同通过 try...catch 捕获错误
// try {
//     var data = fs.readFileSync('01helloWorld.js', 'utf-8');
//     console.log(data);
// } catch (err) {
```

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

```
// 写文件
// 参数 文件路径, 写入的数据, 回调, 默认是 UTF-8,如果传入的数据是 Buffer 则 写入二进制文件
// 文件名不存在会创建该文件, 写入的内容会覆盖掉之前文件的内容
var data = 'hello writeFile asyns';
// fs.writeFile('day01.md', data, err => {
//     if(err) {
//         console.log(err);
//     }else {
//         console.log('ok');
//     }
// })
// 同步写入
// fs.writeFileSync('day01.md', data);
```

```
// 获取文件或目录 的详细信息
// try {
//     var stat = fs.statSync('sample.txt')
//     // 是否是文件:
//     console.log('isFile: ' + stat.isFile());
//     // 是否是目录:
//     console.log('isDirectory: ' + stat.isDirectory());
//     if (stat.isFile()) {
//         // 文件大小:
//         console.log('size: ' + stat.size);
//         // 创建时间, Date 对象:
//         console.log('birth time: ' + stat.birthtime);
//         // 修改时间, Date 对象:
//         console.log('modified time: ' + stat.mtime);
//     }
// } catch (err) {
//     console.log(err);
// }
```

```
'use strict';

var fs = require('fs');
// 参数:  文件路径,  编码格式,  处理函数
// fs.readFile('01helloWorld.js', 'utf-8', function(err, data) {
//     if(err) {
//         console.log(err);
```

```
//     }else {
//         console.log(data);
//     }
// })
```

```
// 获取二进制文件
// fs.readFile('01.docx', (err, data) => {
//     if(err) {
//         console.log(err);
//     }else {
//         console.log(data);
//         console.log(data.length + 'bytes');
//         // 二进制文件返回的 data 是一个 Buffer 对象，可以转换成 String 对象
//         let text = data.toString('utf-8');
//         console.log(text);
//         // 可以将 String 转换成 Buffer 对象
//         let buf = Buffer.from(text, 'utf-8');
//         console.log(buf);
//     }
// })
```

```
// 同步读文件， 同步读取的函数和异步相比，多了一个 Sync 后缀，并且不接受回调函数，函数直接返回结果
// var data = fs.readFileSync('01.helloWorld.js', 'utf-8');
// console.log(data);
// 发生错误需要同通过 try...catch 捕获错误
// try {
//     var data = fs.readFileSync('01helloWorld.js', 'utf-8');
//     console.log(data);
// } catch (err) {
//     console.log(err);
// }
```

```
// 写文件
// 参数 文件路径， 写入的数据， 回调， 默认是 UTF-8，如果传入的数据是 Buffer 则 写入二进制文件
// 文件名不存在会创建该文件，写入的内容会覆盖掉之前文件的内容
var data = 'hello writeFile asyns';
// fs.writeFile('day01.md', data, err => {
//     if(err) {
//         console.log(err);
//     }else {
//         console.log('ok');
//     }
```

```
// })
// 同步写入
// fs.writeFileSync('day01.md', data);
```

```
// 获取文件或目录 的详细信息
// try {
//     var stat = fs.statSync('sample.txt')
//     // 是否是文件:
//     console.log('isFile: ' + stat.isFile());
//     // 是否是目录:
//     console.log('isDirectory: ' + stat.isDirectory());
//     if (stat.isFile()) {
//         // 文件大小:
//         console.log('size: ' + stat.size);
//         // 创建时间, Date 对象:
//         console.log('birth time: ' + stat.birthtime);
//         // 修改时间, Date 对象:
//         console.log('modified time: ' + stat.mtime);
//     }
// } catch (err) {
//     console.log(err);
// }
```

2. Stream

```
'use strict';
// 流分为标准输入流（stdin), 标准输出流(stdout)

// 文件流读取文本内容
var fs = require('fs');
```

```
// // 打开一个流
// var rs = fs.createReadStream('day01.md', 'utf-8');
```

```
// // 监听 data 事件，会一点一点的获取流里的内容
// rs.on('data', function(chunk) {
//     console.log('DATA');
//     console.log(chunk);

// })
```

```
// // 这个流读取到末尾了
// rs.on('end', function(){
//     console.log('END');
// })
// // 这个流出错误了
// rs.on('error', function(err) {
//     console.log('ERROR');
//     console.log(err);
// })
```

```
// 文件流写入数据
// 打开输出流
// var ws1 = fs.createWriteStream('output1.txt', 'utf-8');
// // 写入数据
// ws1.write('使用 Stream 写入文本数据。。。1\n');
// ws1.write('END.');
// // 结束
// ws1.end();
```

```
// var ws1 = fs.createWriteStream('output2.txt', 'utf-8');
// // 写入数据
// ws1.write(Buffer.from('使用 Stream 写入文本数据。。。1\n', 'utf-8'));
// ws1.write(Buffer.from('END.', 'utf-8'));
// // 结束
// ws1.end();
```

```
// 将文件串联起来，复制文件
var rs = fs.createReadStream('03.cmd_1.js');
var ws = fs.createWriteStream('copide.txt');
// 流读取完毕会触发 on 事件，  如果不希望可以传入参数 {end: false}
rs.pipe(ws);
```

五、明天计划

学习 nodejs 4,7,8 章节