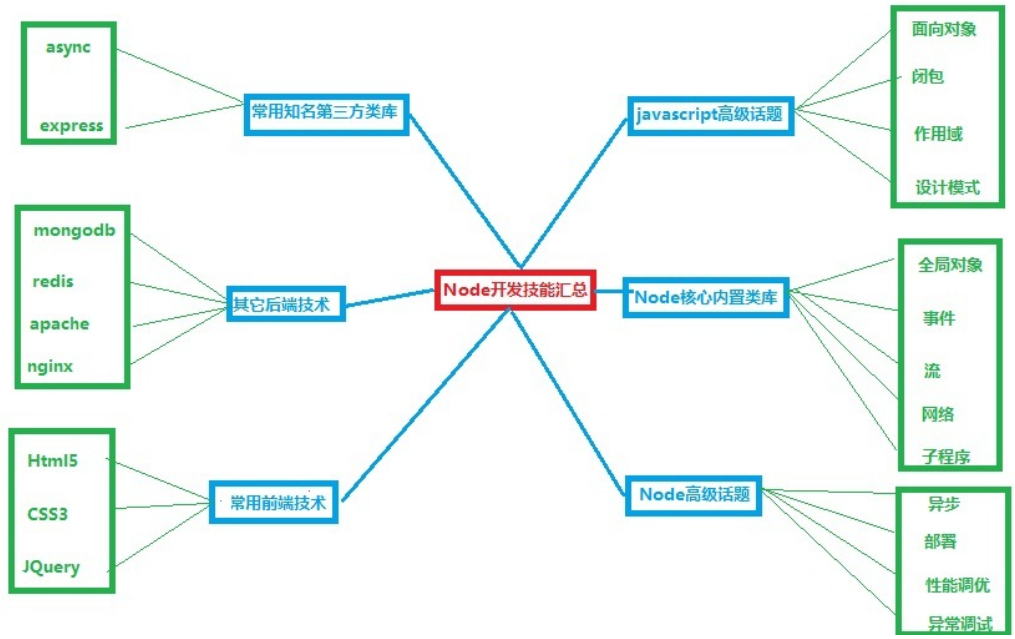
node开发技能图解



起源

- node正风生火起，很多介绍却停留在入门阶段，无法投入生产
- node相关的高质量面试题更是少之又少，很难全面考查应聘者的node能力
- 许多文章在讲第三方类库，可是这些库质量差距较大，一旦遇到问题怎么办
- 必需的，全面了解node核心才能成为一名合格的node开发人员

目标与原则

- 前后端兼顾，更侧重后端
- 理论实战兼顾，侧重考察对实战中应用较多的理论的理解
- 参考答案简单明了，一针见血，不为追求严谨而浪费口舌，绕弯子
- 尽量用代码讲清理论的应用与区别，以接地气
- 终极目标是让大家对node有一个快速完整的认识

内容大纲

- javascript高级话题(面向对象，作用域，闭包，设计模式等)
- node核心内置类库(事件，流，文件，网络等)
- node高级话题(异步，部署，性能调优，异常调试等)
- 常用知名第三方类库(Async, Express等)
- 其它相关后端常用技术(MongoDB, Redis, Apache, Nginx等)
- 常用前端技术(Html5, CSS3, JQuery等)

javascript高级话题(面向对象，作用域，闭包，设计模式等)

- **1. 常用js类定义的方法有哪些？**

参考答案：主要有构造函数原型和对象创建两种方法。原型法是通用老方法，对象创建是ES5推荐使用的方法.目前来看，原型法更普遍.

代码演示 1) 构造函数方法定义类

```
function Person(){
    this.name = 'michaelqin';
}
Person.prototype.sayName = function(){
    alert(this.name);
}

var person = new Person();
person.sayName();
```

2) 对象创建方法定义类

```
var Person = {
    name: 'michaelqin',
    sayName: function(){ alert(this.name); }
};

var person = Object.create(Person);
person.sayName();
```

- **2. js类继承的方法有哪些**

参考答案：原型链法，属性复制法和构造器应用法. 另外，由于每个对象可以是一个类，这些方法也可以用于对象类的继承．

代码演示 1) 原型链法

```
function Animal() {
    this.name = 'animal';
}
Animal.prototype.sayName = {
    alert(this.name);
};

function Person() {}
Person.prototype = Animal.prototype; // 人继承自动物
Person.prototype.constructor = 'Person'; // 更新构造函数为人
```

2) 属性自制法

```
function Animal() {
    this.name = 'animal';
}
Animal.prototype.sayName = {
    alert(this.name);
};

function Person() {}

for(prop in Animal.prototype) {
    Person.prototype[prop] = Animal.prototype[prop];
} // 复制动物的所有属性到人量边
Person.prototype.constructor = 'Person'; // 更新构造函数为人
```

3) 构造器应用法

```
function Animal() {
    this.name = 'animal';
}
Animal.prototype.sayName = {
    alert(this.name);
};

function Person() {
    Animal.call(this); // apply, call, bind方法都可以．细微区别，后面会提到．
}
```

- **3. js类多重继承的实现方法是怎么样的?**

参考答案：就是类继承里边的属性复制法来实现．因为当所有父类的prototype属性被复制后，子类自然拥有类似行为和属性．

- **4. js里的作用域是什么样子的？**

参考答案：大多数语言里边都是块作作用域，以{}进行限定，js里边不是．js里边叫函数作用域，就是一个变量在全函数里有效．比如有个变量p1在函数最后一行定义，第一行也有效，但是值是undefined.

代码演示

```
var globalVar = 'global var';

function test() {
    alert(globalVar); // undefined, 因为globalVar在本函数内被重定义了，导致全局失效，这里使用函数内的变量值，可是此时还没定义
    var globalVar = 'overrided var'; //    globalVar在本函数内被重定义
    alert(globalVar);    // overrided var
}
alert(globalVar); // global var，使用全局变量
```

- **5. js里边的this指的是什么?**

参考答案: this指的是对象本身，而不是构造函数．

代码演示

```
function Person() {
}
Person.prototype.sayName() { alert(this.name); }

var person1 = new Person();
person1.name = 'michaelqin';
person1.sayName(); // michaelqin
```

- **6. apply, call和bind有什么区别?**

参考答案：三者都可以把一个函数应用到其他对象上，注意不是自身对象．apply,call是直接执行函数调用，bind是绑定，执行需要再次调用．apply和call的区别是apply接受数组作为参数，而call是接受逗号分隔的无限多个参数列表，

代码演示

```
function Person() {
}
Person.prototype.sayName() { alert(this.name); }

var obj = {name: 'michaelqin'}; // 注意这是一个普通对象，它不是Person的实例
1) apply
Person.prototype.sayName.apply(obj, [param1, param2, param3]);

2) call
Person.prototype.sayName.call(obj, param1, param2, param3);

3) bind
var sn = Person.prototype.sayName.bind(obj);
sn([param1, param2, param3]); // bind需要先绑定，再执行
sn(param1, param2, param3); // bind需要先绑定，再执行
```

- **7. caller, callee和arguments分别是什么?**

参考答案: caller,callee之间的关系就像是employer和employee之间的关系，就是调用与被调用的关系，二者返回的都是函数对象引用．arguments是函数的所有参数列表，它是一个类数组的变量．

代码演示

```
function parent(param1, param2, param3) {
    child(param1, param2, param3);
}

function child() {
    console.log(arguments); // { '0': 'mqin1', '1': 'mqin2', '2': 'mqin3' }
    console.log(arguments.callee); // [Function: child]
    console.log(child.caller); // [Function: parent]
}

parent('mqin1', 'mqin2', 'mqin3');
```

- **8. 什么是闭包，闭包有哪些用处?**

参考答案: 闭包这个术语，无论中文翻译还是英文解释都太２Ｂ了，我必须骂人，因为它什么其实都不是．非要讲它是什么的话，两个字函数，更多字嵌套函数的父子自我引用关系．所有函数都是闭包．通俗的说，闭包就是作用域范围，因为js是函数作用域，所以函数就是闭包．全局函数的作用域范围就是全局，所以无须讨论．更多的应用其实是在内嵌函数，这就会涉及到内嵌作用域，或者叫作用域链．说到内嵌，其实就是父子引用关系(父函数包含子函数，子函数因为函数作用域又引用父函数，这它妈不是死结吗？所以叫闭包），这就会带来另外一个问题，什么时候引用结束？如果不结束，就会一直占用内存，引起内存泄漏．好吧，不用的时候就引用设为空，死结就解开了．

- **9. defineProperty, hasOwnProperty, isEnumerable都是做什么用的？**

参考答案：Object.defineProperty(obj, prop, descriptor)用来给对象定义属性,有value,writable,configurable,enumerable,set/get等.hasOwnProerty用于检查某一属性是不是存在于对象本身，继承来的父亲的属性不算．isEnumerable用来检测某一属性是否可遍历，也就是能不能用for..in循环来取到.

- **10. js常用设计模式的实现思路，单例，工厂，代理，装饰，观察者模式等**

参考答案：

```
1) 单例：   任意对象都是单例，无须特别处理
var obj = {name: 'michaelqin', age: 30};

2) 工厂: 就是同样形式参数返回不同的实例
function Person() { this.name = 'Person1'; }
function Animal() { this.name = 'Animal1'; }

function Factory() {}
Factory.prototype.getInstance = function(className) {
    return eval('new ' + className + '()');
}

var factory = new Factory();
var obj1 = factory.getInstance('Person');
var obj2 = factory.getInstance('Animal');
console.log(obj1.name); // Person1
console.log(obj2.name); // Animal1

3) 代理: 就是新建个类调用老类的接口,包一下

function Person() { }
```

```
Person.prototype.sayName = function() { console.log('michaelqin'); }
Person.prototype.sayAge = function() { console.log(30); }

function PersonProxy() {
    this.person = new Person();
    var that = this;
    this.callMethod = function(functionName) {
        console.log('before proxy:', functionName);
        that.person[functionName](); // 代理
        console.log('after proxy:', functionName);
    }
}

var pp = new PersonProxy();
pp.callMethod('sayName'); // 代理调用Person的方法sayName()
pp.callMethod('sayAge'); // 代理调用Person的方法sayAge()

4) 观察者: 就是事件模式，比如按钮的onclick这样的应用.
function Publisher() {
    this.listeners = [];
}
Publisher.prototype = {
    'addListener': function(listener) {
        this.listeners.push(listener);
    },

    'removeListener': function(listener) {
        delete this.listeners[listener];
    },

    'notify': function(obj) {
        for(var i = 0; i < this.listeners.length; i++) {
            var listener = this.listeners[i];
            if (typeof listener !== 'undefined') {
                listener.process(obj);
            }
        }
    }
}; // 发布者

function Subscriber() {

}
Subscriber.prototype = {
    'process': function(obj) {
        console.log(obj);
    }
};    // 订阅者


var publisher = new Publisher();
publisher.addListener(new Subscriber());

publisher.addListener(new Subscriber());
```

```
publisher.notify({name: 'michaelqin', ageo: 30}); // 发布一个对象到所有订阅者
publisher.notify('2 subscribers will both perform process'); // 发布一个字符串到所有订阅者
```

- **11. 列举数组相关的常用方法**

参考答案: push/pop, shift/unshift, split/join, slice/splice/concat, sort/reverse, map/reduce, forEach, filter

- **12. 列举字符串相关的常用方法**

参考答案: indexOf/lastIndexOf/charAt, split/match/test, slice/substring/substr, toLowerCase/toUpperCase

node核心内置类库(事件，流，文件，网络等)

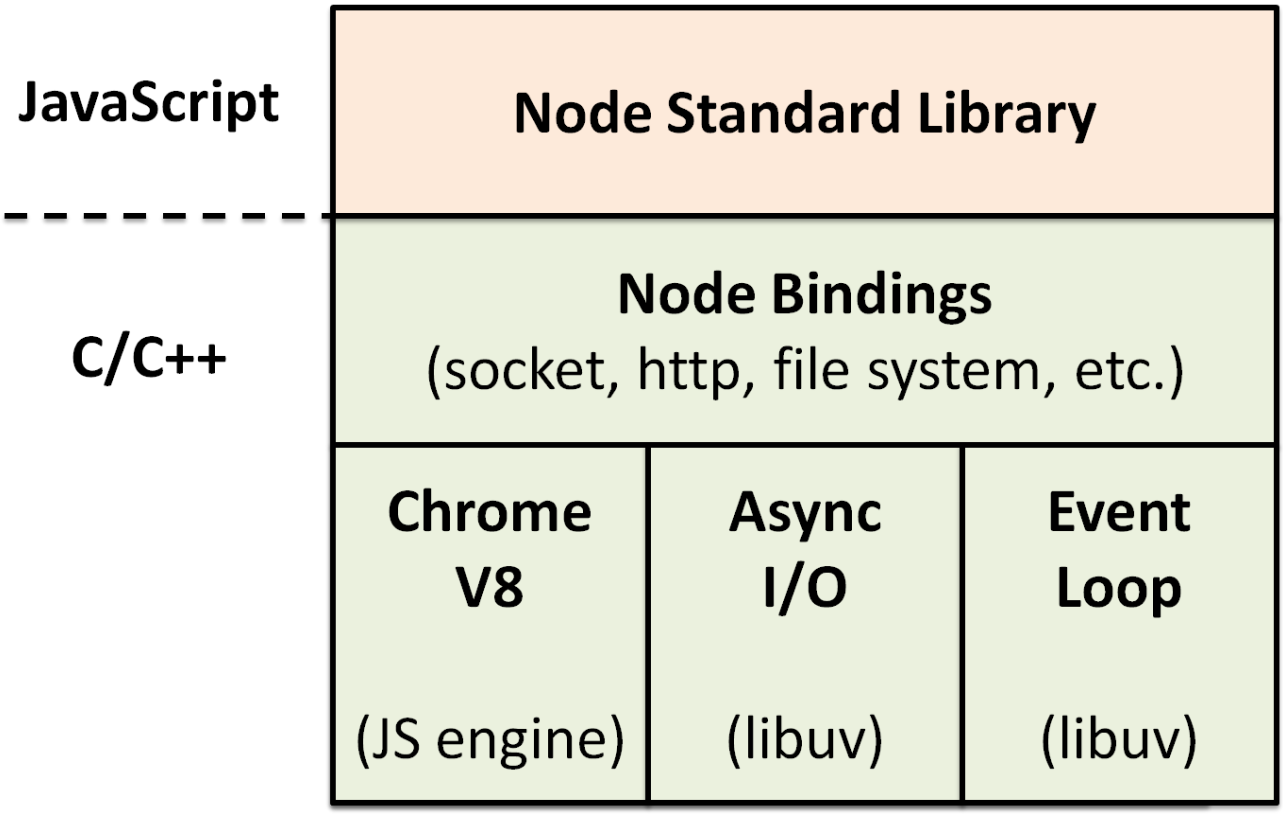
node概览

- **1. 为什么要用node?**

参考答案: 总结起来node有以下几个特点:简单强大，轻量可扩展．简单体现在node使用的是javascript,json来进行编码，人人都会；强大体现在非阻塞IO,可以适应分块传输数据，较慢的网络环境，尤其擅长高并发访问；轻量体现在node本身既是代码，又是服务器，前后端使用统一语言;可扩展体现在可以轻松应对多实例，多服务器架构，同时有海量的第三方应用组件．

- **2. node的构架是什么样子的?**

参考答案: 主要分为三层，应用app >> V8及node内置架构 >> 操作系统. V8是node运行的环境，可以理解为node虚拟机．node内置架构又可分为三层: 核心模块(javascript实现) >> c++绑定 >> libuv + CAes + http.



- **3. node有哪些核心模块?**

参考答案: EventEmitter, Stream, FS, Net和全局对象

node全局对象

- **1. node有哪些全局对象?**

参考答案: process, console, Buffer和exports

- **2. process有哪些常用方法?**

参考答案: process.stdin, process.stdout, process.stderr, process.on, process.env, process.argv, process.arch, process.platform, process.exit

- **3. console有哪些常用方法?**

参考答案: console.log/console.info, console.error/console.warning, console.time/console.timeEnd, console.trace, console.table

- **4. node有哪些定时功能?**

参考答案: setTimeout/clearTimeout, setInterval/clearInterval, setImmediate/clearImmediate, process.nextTick

- **5. node中的事件循环是什么样子的?**

参考答案: event loop其实就是一个事件队列，先加入先执行，执行完一次队列，再次循环遍历看有没有新事件加入队列．执行中的叫IO events, setImmediate是在当前队列立即执行,setTimout/setInterval是把执行定时到下一个队列，process.nextTick是在当前执行完，下次遍历前执行．所以总体顺序是: IO events >> setImmediate >> setTimeout/setInterval >> process.nextTick

- **6. node中的Buffer如何应用?**

参考答案: Buffer是用来处理二进制数据的，比如图片，mp3,数据库文件等.Buffer支持各种编码解码，二进制字符串互转．

EventEmitter

- **1. 什么是EventEmitter?**

参考答案: EventEmitter是node中一个实现观察者模式的类，主要功能是监听和发射消息，用于处理多模块交互问题.

- **2. 如何实现一个EventEmitter?**

参考答案: 主要分三步：定义一个子类，调用构造函数，继承EventEmitter

代码演示

```
var util = require('util');
var EventEmitter = require('events').EventEmitter;

function MyEmitter() {
    EventEmitter.call(this);
} // 构造函数

util.inherits(MyEmitter, EventEmitter); // 继承


var em = new MyEmitter();
```

```
em.on('hello', function(data) {
    console.log('收到事件hello的数据:', data);
}); // 接收事件，并打印到控制台
em.emit('hello', 'EventEmitter传递消息真方便!');
```

- **3. EventEmitter有哪些典型应用?**

参考答案: 1) 模块间传递消息 2) 回调函数内外传递消息 3) 处理流数据，因为流是在EventEmitter基础上实现的. 4) 观察者模式发射触发机制相关应用

- **4. 怎么捕获EventEmitter的错误事件?**

参考答案: 监听error事件即可．如果有多个EventEmitter,也可以用domain来统一处理错误事件.

代码演示

```
var domain = require('domain');
var myDomain = domain.create();
myDomain.on('error', function(err){
    console.log('domain接收到的错误事件:', err);
}); // 接收事件并打印
myDomain.run(function(){
    var emitter1 = new MyEmitter();
    emitter1.emit('error', '错误事件来自emitter1');
    emitter2 = new MyEmitter();
    emitter2.emit('error', '错误事件来自emitter2');
});
```

- **5. EventEmitter中的newListenser事件有什么用处?**

参考答案: newListener可以用来做事件机制的反射，特殊应用，事件管理等．当任何on事件添加到EventEmitter时，就会触发newListener事件，基于这种模式，我们可以做很多自定义处理.

代码演示

```
var emitter3 = new MyEmitter();
emitter3.on('newListener', function(name, listener) {
    console.log("新事件的名字:", name);
    console.log("新事件的代码:", listener);
    setTimeout(function(){ console.log("我是自定义延时处理机制"); }, 1000);
});
emitter3.on('hello', function(){
    console.log('hello    node');
});
```

Stream

- **1. 什么是Stream?**

参考答案: stream是基于事件EventEmitter的数据管理模式．由各种不同的抽象接口组成，主要包括可写，可读，可读写，可转换等几种类型．

- **2. Stream有什么好处?**

参考答案: 非阻塞式数据处理提升效率，片断处理节省内存，管道处理方便可扩展等.

- **3. Stream有哪些典型应用?**

参考答案: 文件，网络，数据转换，音频视频等.

- **4. 怎么捕获Stream的错误事件?**

参考答案: 监听error事件，方法同EventEmitter.

- **5. 有哪些常用Stream,分别什么时候使用?**

参考答案: Readable为可被读流，在作为输入数据源时使用；Writable为可被写流,在作为输出源时使用；Duplex为可读写流,它作为输出源接受被写入，同时又作为输入源被后面的流读出．Transform机制和Duplex一样，都是双向流，区别时Transfrom只需要实现一个函数*transfrom(chunk, encoding, callback);而Duplex需要分别实现* read(size)函数和_write(chunk, encoding, callback)函数.

- **6. 实现一个Writable Stream?**

参考答案: 三步走:1)构造函数call Writable 2)　继承Writable 3) 实现_write(chunk, encoding, callback)函数

代码演示

```
var Writable = require('stream').Writable;
var util = require('util');

function MyWritable(options) {
    Writable.call(this, options);
} // 构造函数
util.inherits(MyWritable, Writable); // 继承自Writable
MyWritable.prototype._write = function(chunk, encoding, callback) {
    console.log("被写入的数据是:", chunk.toString()); // 此处可对写入的数据进行处理
    callback();
};

process.stdin.pipe(new MyWritable()); // stdin作为输入源，MyWritable作为输出源
```

文件系统

- **1. 内置的fs模块架构是什么样子的?**

参考答案: fs模块主要由下面几部分组成: 1) POSIX文件Wrapper,对应于操作系统的原生文件操作 2) 文件流 fs.createReadStream和fs.createWriteStream 3) 同步文件读写,fs.readFileSync和fs.writeFileSync 4) 异步文件读写, fs.readFile和fs.writeFile

- **2. 读写一个文件有多少种方法?**

参考答案: 总体来说有四种: 1) POSIX式低层读写 2) 流式读写 3) 同步文件读写 4) 异步文件读写

- **3. 怎么读取json配置文件?**

参考答案: 主要有两种方式，第一种是利用node内置的require('data.json')机制，直接得到js对象; 第二种是读入文件入内容，然后用JSON.parse(content)转换成js对象．二者的区别是require机制情况下，如果多个模块都加载了同一个json文件，那么其中一个改变了js对象，其它跟着改变，这是由node模块的缓存机制造成的，只有一个js模块对象; 第二种方式则可以随意改变加载后的js变量，而且各模块互不影响，因为他们都是独立的，是多个js对象.

- **4. fs.watch和fs.watchFile有什么区别，怎么应用?**

参考答案: 二者主要用来监听文件变动．fs.watch利用操作系统原生机制来监听，可能不适用网络文件系统; fs.watchFile则是定期检查文件状态变更，适用于网络文件系统，但是相比fs.watch有些慢，因为不是实时机制．

网络

- **1. node的网络模块架构是什么样子的?**

参考答案: node全面支持各种网络服务器和客户端，包括tcp, http/https, tcp, udp, dns, tls/ssl等.

- **2. node是怎样支持https,tls的?**

参考答案: 主要实现以下几个步骤即可: 1) openssl生成公钥私钥 2) 服务器或客户端使用https替代http 3) 服务器或客户端加载公钥私钥证书

- **3. 实现一个简单的http服务器?**

参考答案: 经典又很没毛意义的一个题目．思路是加载http模块，创建服务器，监听端口.

代码演示

```
var http = require('http'); // 加载http模块

http.createServer(function(req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'}); // 200代表状态成功, 文档类型是给浏览器识别用的
    res.write('<meta charset="UTF-8"> <h1>我是标题啊！</h1> <font color="red">这么原生，初级的服务器，下辈子能用着吗?!</font>'); // 返回给客户端的html数据
    res.end(); // 结束输出流
}).listen(3000); // 绑定3ooo, 查看效果请访问 http://localhost:3000
```

child-process

- **1. 为什么需要child-process?**

参考答案: node是异步非阻塞的，这对高并发非常有效．可是我们还有其它一些常用需求，比如和操作系统shell命令交互，调用可执行文件，创建子进程进行阻塞式访问或高CPU计算等，child-process就是为满足这些需求而生的．child-process顾名思义，就是把node阻塞的工作交给子进程去做．

- **2. exec,execFile,spawn和fork都是做什么用的?**

参考答案: exec可以用操作系统原生的方式执行各种命令，如管道 cat ab.txt | grep hello; execFile是执行一个文件; spawn是流式和操作系统进行交互; fork是两个node程序(javascript)之间时行交互.

- **3. 实现一个简单的命令行交互程序?**

参考答案: 那就用spawn吧.

代码演示

```
var cp = require('child_process');

var child = cp.spawn('echo', ['你好', "钩子"]); // 执行命令
child.stdout.pipe(process.stdout); // child.stdout是输入流，process.stdout是输出流
// 这句的意思是将子进程的输出作为当前程序的输入流，然后重定向到当前程序的标准输出，即控制台
```

- \4. 两个node程序之间怎样交互?

参考答案: 用fork嘛，上面讲过了．原理是子程序用process.on, process.send，父程序里用child.on,child.send进行交互. 代码演示

```
1) fork-parent.js
var cp = require('child_process');
var child = cp.fork('./fork-child.js');
child.on('message', function(msg){
    console.log('老爸从儿子接受到数据:', msg);
});
child.send('我是你爸爸，送关怀来了!');

2) fork-child.js
process.on('message', function(msg){
    console.log("儿子从老爸接收到的数据:", msg);
    process.send("我不要关怀，我要银民币！");
});
```

- **5. 怎样让一个js文件变得像linux命令一样可执行?**

参考答案: 1) 在myCommand.js文件头部加入 #!/usr/bin/env node 2) chmod命令把js文件改为可执行即可 3) 进入文件目录，命令行输入myComand就是相当于node myComand.js了

- **6. child-process和process的stdin,stdout,stderror是一样的吗?**

参考答案: 概念都是一样的，输入，输出，错误，都是流．区别是在父程序眼里，子程序的stdout是输入流，stdin是输出流．

node高级话题(异步，部署，性能调优，异常调试等)

- **1. node中的异步和同步怎么理解**

参考答案: node是单线程的，异步是通过一次次的循环事件队列来实现的．同步则是说阻塞式的IO,这在高并发环境会是一个很大的性能问题，所以同步一般只在基础框架的启动时使用，用来加载配置文件，初始化程序什么的．

- **2. 有哪些方法可以进行异步流程的控制?**

参考答案: 1) 多层嵌套回调 2)　为每一个回调写单独的函数，函数里边再回调 3) 用第三方框架比方async, q, promise等

- **3. 怎样绑定node程序到80端口?**

参考答案: 多种方式 1) sudo 2) apache/nginx代理 3) 用操作系统的firewall iptables进行端口重定向

- **4. 有哪些方法可以让node程序遇到错误后自动重启?**

参考答案: 1) runit 2) forever 3) nohup npm start &

- **5. 怎样充分利用多个CPU?**

参考答案: 一个CPU运行一个node实例

- **6. 怎样调节node执行单元的内存大小?**

参考答案: 用--max-old-space-size 和 --max-new-space-size 来设置 v8 使用内存的上限

- **7. 程序总是崩溃，怎样找出问题在哪里?**

参考答案: 1) node --prof 查看哪些函数调用次数多 2) memwatch和heapdump获得内存快照进行对比，查找内存溢出

- **8. 有哪些常用方法可以防止程序崩溃?**

参考答案: 1) try-catch-finally 2) EventEmitter/Stream error事件处理 3) domain统一控制 4) jshint静态检查 5) jasmine/mocha进行单元测试

- **9. 怎样调试node程序?**

参考答案: node --debug app.js 和node-inspector

常用知名第三方类库(Async, Express等)

- **1. async都有哪些常用方法，分别是怎么用?**

参考答案: async是一个js类库，它的目的是解决js中异常流程难以控制的问题．async不仅适用在node.js里，浏览器中也可以使用． 1) async.parallel并行执行完多个函数后，调用结束函数

```
async.parallel([
    function(){ ... },
    function(){ ... }
], callback);
```

2) async.series串行执行完多个函数后，调用结束函数

```
async.series([
    function(){ ... },
    function(){ ... }
]);
```

3) async.waterfall依次执行多个函数，后一个函数以前面函数的结果作为输入参数

```
  async.waterfall([
    function(callback) {
      callback(null, 'one', 'two');
    },
    function(arg1, arg2, callback) {
     // arg1 now equals 'one' and arg2 now equals 'two'
      callback(null, 'three');
    },
    function(arg1, callback) {
      // arg1 now equals 'three'
```

```
            callback(null, 'done');
        }
    ], function (err, result) {
        // result now equals 'done'
    });
```

4) async.map异步执行多个数组，返回结果数组

```
async.map(['file1','file2','file3'], fs.stat, function(err, results){
    // results is now an array of stats for each file
});
```

5) async.filter异步过滤多个数组，返回结果数组

```
async.filter(['file1','file2','file3'], fs.exists, function(results){
    // results now equals an array of the existing files
});
```

- **2. express项目的目录大致是什么样子的**

参考答案: app.js, package.json, bin/www, public, routes, views.

- **3. express常用函数**

参考答案: express.Router路由组件,app.get路由定向，app.configure配置，app.set设定参数,app.use使用中间件

- **4. express中如何获取路由的参数**

参考答案: /users/:name使用req.params.name来获取; req.body.username则是获得表单传入参数username; express路由支持常用通配符 ?, +, *, and ()

- **5. express response有哪些常用方法**

参考答案: res.download() 弹出文件下载 res.end() 结束response res.json() 返回json res.jsonp() 返回jsonp res.redirect() 重定向请求 res.render() 渲染模板 res.send() 返回多种形式数据 res.sendFile 返回文件 res.sendStatus() 返回状态

其它相关后端常用技术(MongoDB, Redis, Apache, Nginx等)

- **1. mongodb有哪些常用优化措施**

参考答案: 类似传统数据库，索引和分区．

- **2. redis支持哪些功能**

参考答案: set/get, hset/hget, publish/subscribe, expire

- **3. redis最简单的应用**

参考答案:

```
var redis = require("redis"),
    client = redis.createClient();

client.set("foo_rand000000000000", "some fantastic value");
client.get("foo_rand000000000000", function (err, reply) {
    console.log(reply.toString());
});
client.end();
```

- **4. apache,nginx有什么区别?**

参考答案: 二者都是代理服务器，功能类似．apache应用简单，相当广泛．nginx在分布式，静态转发方面比较有优势．

常用前端技术(Html5, CSS3, JQuery等)

- **1. Html5有哪些比较实用新功能**

参考答案: File API支持本地文件操作; Canvans/SVG支持绘图; 拖拽功能支持; 本地存储支持; 表单多属性验证支持; 原生音频视频支持等

- **2. CSS3/JQuery有哪些学常见选择器**

参考答案: id, 元素，属性, 值，父子兄弟, 序列等

- **3. JQuery有哪些经典应用**

参考答案: 文档选择，文档操作，动画, ajax, json, js扩展等.

对Node 的优点和缺点提出了自己的看法：

- （优点）因为Node 是基于事件驱动和无阻塞的，所以非常适合处理并发请求， 因此构建在Node 上的代理服务器相比其他技术实现（如Ruby）的服务器表现要好得多。 此外，与Node 代理服务器交互的客户端代码是由javascript 语言编写的， 因此客户端和服务器端都用同一种语言编写，这是非常美妙的事情。
- （缺点）Node 是一个相对新的开源项目，所以不太稳定，它总是一直在变， 而且缺少足够多的第三方库支持。看起来，就像是Ruby/Rails 当年的样子。
    1. 需求：实现一个页面操作不会整页刷新的网站，并且能在浏览器前进、后退 时正确响应。给出你的技术实现方案？ 至少给出自己的思路（url-hash,可以使用已有的一些框架history.js 等）
        1. Node.js 的适用场景？
           1)、实时应用：如在线聊天，实时通知推送等等（如socket.io）
           2)、分布式应用：通过高效的并行I/O 使用已有的数据
           3)、工具类应用：海量的工具，小到前端压缩部署（如grunt），大到桌面图形界面应用程序
           4)、游戏类应用：游戏领域对实时和并发有很高的要求（如网易的pomelo 框架）
           5)、利用稳定接口提升Web 渲染能力
           6)、前后端编程语言环境统一：前端开发人员可以非常快速地切入到服务器端的开发（如著名的纯Javascript 全栈式MEAN 架构）

1. (如果会用node) 知道route, middleware, cluster, nodemon, pm2,
   server-side rendering 么?
   Nodejs 相关概念的理解程度

2. 解释一下Backbone 的MVC 实现方式？
   流行的MVC 架构模式
3. 什么是“前端路由”?什么时候适合使用“前端路由”? “前端路由”有哪
   些优点和缺点?
   熟悉前后端通信相关知识

对Node 的优点和缺点提出了自己的看法？

优点：

- 因为Node 是基于事件驱动和无阻塞的，所以非常适合处理并发请求，因此构建在Node 上的代理服务器相比其他技术实现（如Ruby）的服务器表现要好得多。
- 与Node 代理服务器交互的客户端代码是由javascript 语言编写的，因此客户端和服务器端都用同一种语言编写，这是非常美妙的事情。
  缺点：
- Node 是一个相对新的开源项目，所以不太稳定，它总是一直在变。
- 缺少足够多的第三方库支持。看起来，就像是Ruby/Rails 当年的样子（第三方库现在已经很丰富了，所以这个缺点可以说不存在了）。