## 第一讲：

### 1、箭头函数有哪些特性

1. 箭头函数是匿名函数，不绑定自己的this,arguments,super,new.target
2. 箭头函数会捕获其所在上下文的this值，作为自己的this值，在使用call/apply绑定时，相当于只是传入了参数，对this没有影响
3. 箭头函数不绑定arguments，取而代之用rest参数…解决
4. 箭头函数当方法使用的时候，没有定义this绑定
5. 箭头函数不能作为构造函数，和 new 一起用就会抛出错误
6. 箭头函数没有原型属性
7. 不能简单返回对象字面量

//1相比普通函数更简洁的语法

2本身没有this，捕获其所在上下文（作用域）的this值，作为自己的 this 值

3不能使用new

4不绑定arguments，用rest参数...解决

5使用call()和apply()调用

6箭头函数没有原型属性

7不能简单返回对象字面量

8箭头函数不能当做Generator函数,不能使用yield关键字

### 2、let和var的区别

1. 注意块级作用域
2. 暂时性死区
3. 不存在变量提升
4. 同一作用域内不允许重复

### 实现数组过滤重复（方式不限）

var arr = [1, 2, 2, 1, 3];

        var obj = {};

       1 arr.forEach(function(val, index) {

            obj[val] === arr[index];

        });

        console.log(object.keys(obj));

2方法二

//let r=arr.filter(function(value,index,self){

return self.indexOf(index)===value

}

Console.log(r)

3方法三

//let newArr=new Set(arr);

Console.log(newArr)

4方法四

利用sort排序后，相邻元素进行对比

function unique(arr) {

arr.sort( (a,b) => a-b);

for(let i=0; i< arr.length-1; i++){

if(arr[i] === arr[i+1]) {

arr.splice(i+1,1);

}

}

return arr;

}

方法五

for循环嵌套for循环

function unique(arr){

for(let i = 0; i<arr.length; i++) {

for(let j = i+1; j < arr.length; j++) {

if(arr[i] === arr[j]){

arr.splice(j,1); // 去除重复的这个

j--;

}

}

}

return arr;

}

————————————————

### 使用JSON方式实现深拷贝

var obj = { name: '中公教育', title: '教育专家' };

        var obj\_str = JSON.stringify(obj);

        var obj\_clone = JSON.parse(obj\_str);

        obj.name = '我的中公';

        console.log(obj);

        console.log(obj\_clone);

### ES6模板字符串的好处有哪些

1. 模板字符串为构造多行字符串和字符串拼接带来了更加方便的方式。
2. 同一个模板字符串可以通过使用不同的标签，对模板进行进一步处理，输出不同的结果

### 关于对象扩展的属性简写和属性名表达式写法，请各举一个例子

属性名简写：

var title = 'offcn';

        var size = 10;

        var obj = {

            title,

            size,

        };

属性名表达式：

var title = 'offcn';

        obj[title] = '中公教育';

## 第二讲：

### 1、 手写一个promise

var p = new Promise((resolve,reject)={

if(1){

resolve(‘成功’)

}

else{

reject(‘失败’);

}

})

p.then((res)=>{}).catch((err)=>{})

### 2、 class定义类

class Aniaml{

static Type = ‘动物’

constructor(newName){

this.name = newName

}

eat(){conosle.log(‘吃东西’)}

static sayHi(){

console.log(‘大家好’)

}

}

### 3、 class实现继承

class Tigger extends Animal{

constructor(props){

super(props)

}

}

4、谈一谈你对async...await的理解  
 async...await是基于promise的generator语法糖，它用来等待promise的执行结果，常规函数使用await没有效果；async修饰的函数内部return不会得到预期的结果，会得到一个promise对象；await等待的promise结果是resolve状态的内容，reject状态的内容需要使用try...catch获取，await关键字必须要出现在async修饰的函数中，否则报错。

### 5、ES6中的Symbol类型什么作用

ES6新增了Symbol数据类型，它用来生成一个独一无二的值，它Symbol数据常用来给对象属性赋值，让对象属性具备唯一性，不容易被覆盖。

### class 类中的super有哪些用法

Super在类中有两种用法，一个是super方法，一个是super对象；super方法只能出现在constructor方法中，super对象一般出现在子类覆盖父类的方法中。

### 静态和成员的区别

静态的变量或方法归类所有，全局独一份，成员的变量或方法归对象所有，每次实例化对象成员方法和成员变量就会得到一份拷贝；对于一些通用性的属性或方法，可以考虑设置为静态。

### 9、谈一谈你对promise的理解

Promise用来解决异步回调问题，由于js是单线程的，很多异步操作都是依靠回调方法实现的，这种做法在逻辑比较复杂的回调嵌套中会相当复杂；也叫做回调地狱；promise用来将这种繁杂的做法简化，让程序更具备可读性，可维护性；promise内部有三种状态，pedding，fulfilled，rejected；pedding表示程序正在执行但未得到结果，即异步操作没有执行完毕，fulfilled表示程序执行完毕，且执行成功，rejected表示执行完毕但失败；这里的成功和失败都是逻辑意义上的；并非是要报错。其实，promise和回调函数一样，都是要解决数据的传递和消息发送问题，promise中的then一般对应成功后的数据处理，catch一般对应失败后的数据处理。

## 第三讲：

### 深拷贝的原理

Js的深拷贝发生在对象的赋值上，对象赋值的时候仅仅是一个引用赋值，也就是说两个不同变量名指向的是同一份内存空间；基本数据类型则不存在这样的行为；要完成对象的深拷贝需要使用递归遍历所有对象的属性进行赋值，也可以使用JSON.stringfy和JSON.parse操作。

### Set和map的特点

Set是一种类似数组的集合类型，它与数组不同的是，不允许存在重复数据；常用操作方法有：add,delete,has,clear等；遍历使用forEach；

Map是一种类似对象的集合类型，它与对象不同的是，key可以接受对象类型，常用的操作方法有：set,get,has,delete等；遍历使用forEach

### 使用正则实现去除字符串首位空格

var osfipin= '       http://www.cnblogs.com/osfipin/  ';

  osfipin.replace(/(^\s\*)|(\s\*$)/g, "");

## 第四讲：

### node的特点是什么

1）它是一个Javascript运行环境

2）依赖于Chrome V8引擎进行代码解释

3）事件驱动

4）非阻塞I/O

5）轻量、可伸缩，适于实时数据交互应用

6）单进程，单线程

### Module.exports和exports的区别，请用代码示范。

Exports是对Module.exports的引用。

var module = {

            exports: {},

        };

        var exports = module .exports;

        exports.title = '中公教育';

        //此时module.exports新增title属性为中公教育

        exports = { name: '中公教育' };

        //此时module.exports保持原有的内容不变

### http请求和响应的报文格式和常见http状态码

HTTP 请求报文由请求行、请求头部、空行 和 请求包体 4 个部分组成，如下图所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 请求方法 | 空格 | URL | 空格 | 协议版本 | 回车符 | 换行符 | 请求行 |
| 头部字段名称 | :(冒号) | 值 | | | 回车符 | 换行符 | 请求头 |
| ... | | | | | | |
| 头部字段名称 | :(冒号) | 值 | | | 回车符 | 换行符 |
| 空行（回车符或换行符） | | | | | | | 空行 |
| 请求主体 | | | | | | | 请求主体 |

HTTP 响应报文由状态行、响应头部、空行 和 响应包体 4 个部分组成，如下图所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 协议版本 | 空格 | 状态码 | 空格 | 状态码描述 | 回车符 | 换行符 | 状态行 |
| 头部字段名称 | :(冒号) | 值 | | | 回车符 | 换行符 | 响应头 |
| ... | | | | | | |
| 头部字段名称 | :(冒号) | 值 | | | 回车符 | 换行符 |
| 空行（回车符或换行符） | | | | | | | 空行 |
| 响应主体 | | | | | | | 响应主体 |

 状态码由三位数字组成，第一位数字表示响应的类型，常用的状态码有五大类如下所示：

　　100：表示服务器已接收了客户端请求，客户端可继续发送请求;

200：ok//客户端请求成功

　　300：表示服务器要求客户端重定向;

　　404：表示客户端的请求有非法内容;

500：表示服务器未能正常处理客户端的请求而出现意外错误;

2开头 （请求成功）表示成功处理了请求的状态代码。

200   （成功）  服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。

201   （已创建）  请求成功并且服务器创建了新的资源。

202   （已接受）  服务器已接受请求，但尚未处理。

203   （非授权信息）  服务器已成功处理了请求，但返回的信息可能来自另一来源。

204   （无内容）  服务器成功处理了请求，但没有返回任何内容。

205   （重置内容） 服务器成功处理了请求，但没有返回任何内容。

206   （部分内容）  服务器成功处理了部分 GET 请求。

3开头 （请求被重定向）表示要完成请求，需要进一步操作。 通常，这些状态代码用来重定向。

300   （多种选择）  针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。

301   （永久移动）  请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。

302   （临时移动）  服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

303   （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。

304   （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。

305   （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。

307   （临时重定向）  服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

4开头 （请求错误）这些状态代码表示请求可能出错，妨碍了服务器的处理。

400   （错误请求） 服务器不理解请求的语法。

401   （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

403   （禁止） 服务器拒绝请求。

404   （未找到） 服务器找不到请求的网页。

405   （方法禁用） 禁用请求中指定的方法。

406   （不接受） 无法使用请求的内容特性响应请求的网页。

407   （需要代理授权） 此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。

408   （请求超时）  服务器等候请求时发生超时。

409   （冲突）  服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。

410   （已删除）  如果请求的资源已永久删除，服务器就会返回此响应。

411   （需要有效长度） 服务器不接受不含有效内容长度标头字段的请求。

412   （未满足前提条件） 服务器未满足请求者在请求中设置的其中一个前提条件。

413   （请求实体过大） 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。

414   （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。

415   （不支持的媒体类型） 请求的格式不受请求页面的支持。

416   （请求范围不符合要求） 如果页面无法提供请求的范围，则服务器会返回此状态代码。

417   （未满足期望值） 服务器未满足"期望"请求标头字段的要求。

5开头（服务器错误）这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。

500   （服务器内部错误）  服务器遇到错误，无法完成请求。

501   （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。

502   （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。

503   （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。

504   （网关超时）  服务器作为网关或代理，但是没有及时从上游服务器收到请求。

505   （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。

### MIME类型有什么作用

MIME的全称是Multipurpose Internet Mail Extensions，即多用途互联网邮件扩展类型。这是HTTP协议中用来定义文档性质及格式的标准。服务器通过MIME告知响应内容类型，而浏览器则通过MIME类型来确定如何处理文档。

常见MIME类型：

application/octet-stream 未知的应用程序文件

application/json json数据

text/plain 未知的文本文件（纯文本文件），浏览器会认为这是可以直接显示的

注: 浏览器分辨文件是基于MIME的，而不会把文件打开查看是否是其他类型，例如用link链接了一个标记为text/plain的文件，就算此文件是css文件，就算其后缀是.css,浏览器依旧不认

text/css css文件

text/html HTML文件

image/gif gif文件

常见的还有image/jpeg、image/png、image/svg

很多浏览器支持icon image/x-icon

## 第五讲：

### 1、模块加载机制

Nodejs遵循commonjs规范的模块加载机制，使用require加载文件，使用 exports或module.exports导出文件

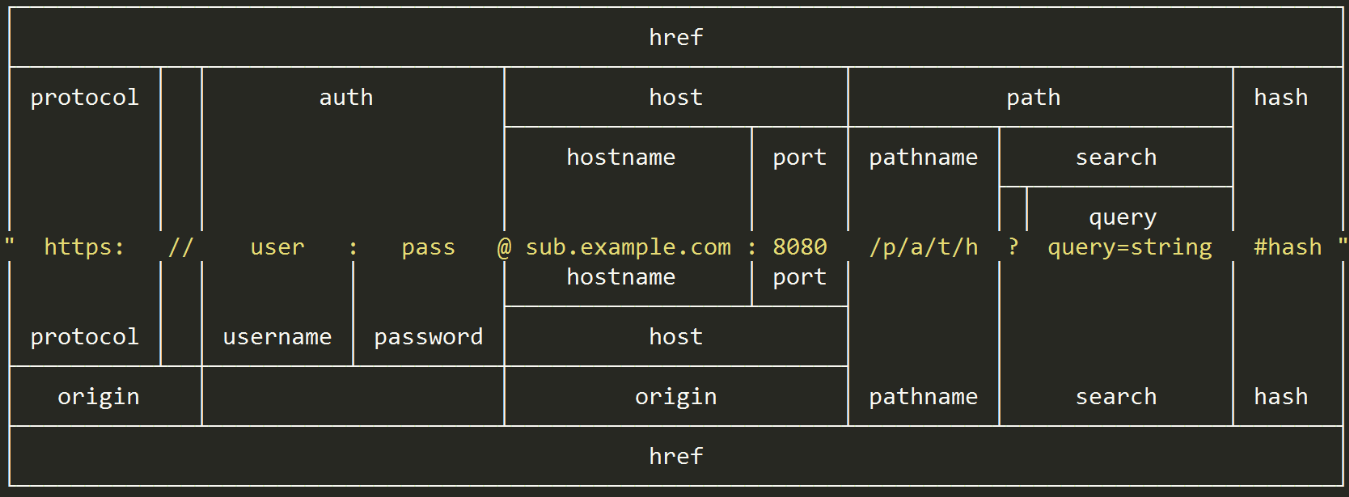
2、NPM的作用

1）允许用户从NPM服务器下载别人编写的第三方包到本地使用

2）允许用户从NPM服务器下载并安装别人编写的命令行程到本地使用

3）允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用

### 2、url的组成：（给出协议+域名+端口+pathname+search+hash即可）



### 3、常用npm指令有哪些：

npm install jquery，npm uninstall jquery，npm search jquery，npm view jquery versions，npm init，npm init -y；

npm config get registry;

npm config set registry <https://registry.npm.taobao.org>

### fs文件操作模块是否能够删除一个非空目录，如果不能应该怎么做

不能，需要递归遍历删除：

function deleteDir(curpath) {

    // 判断当前路径或文件是否存在，存在则进行下一步

    if (fs.existsSync(curpath)) {

        // 读取当前目录下的所有文件或文件夹

        let files = fs.readdirSync(curpath);

        // 循环遍历文件或目录

        files.forEach(val => {

            // './demo'+val

            let tmp\_path = path.join(curpath, val);

            // 判断是目录还是文件

            let isDir = fs.statSync(tmp\_path).isDirectory();

            if (isDir) {

                deleteDir(tmp\_path);

            } else {

                // 同步删除文件

                fs.unlinkSync(tmp\_path);

            }

        });

        // 当前目录遍历完毕后，目录中不存在任何文件，所以删除当前目录

        fs.rmdirSync(curpath);

    }

}

deleteDir('./demo');

function deleteFolderRecursive(path) {

if( fs.existsSync(path) ) {

fs.readdirSync(path).forEach(function(file) {

var curPath = path + "/" + file;

if(fs.statSync(curPath).isDirectory()) { // recurse deleteFolderRecursive(curPath);

} else { // delete file fs.unlinkSync(curPath);

}

});

fs.rmdirSync(path);

}

};

## 第6讲

### 1. express中Router的作用?, 以及Router要解决的问题?

参考答案:

(1): express.Router 可以认为是一个微型的只用来处理中间件与控制器的 app，

它拥有和 app 类似的方法，例如 get、post、all、use 等等。

(2): router它解决了直接把 app 暴露给其它模块使得 app 有被滥用的风险

### 2. npm start 命令, 应该在哪里配置, 在哪里可以正确运行?

参考答案:

(1), 在package.json中的 script中 配置npm 使用的命令

key是npm的自定义的命令名称

value是真正在终端中执行的命令

(2), 在使用npm 运行自定义命令时, 需要确保命令路径的位置上拥有package.json文件

### 3. express生成器的作用是什么?

参考答案: 通过应用生成器工具 express-generator 可以快速创建一个应用的骨架, 包含一整套配置好的服务器配置, 文件和文件夹等, 包括静态资源的暴露等, 包括路由的配置, 和模板引擎配置, 以及404的处理

### 请回答, 前端Ajax请求后端接口/api/getNews, 接口过程中, 都发生了什么?

参考答案: 首先前端代码运行在浏览器中, 由浏览器的XMLHTTPRequest发起一个请求, 如果通过了跨域的检测, 进行http连接, 发送请求信息, 服务器触发对应监听的"/api/getNews"接口的回调函数, 执行其中的代码, 最后res返回响应的内容等给本次请求的地方.

### 5. 请求对象中包含那些内容?

参考答案: 请求协议, 请求方式, 请求的URL, 请求头, 请求体

### 6. 响应对象中包含那些内容?

参考答案: 响应协议, 响应状态码, 响应状态信息, 响应头, 响应的正文数据

### 7. express优点是什么?

参考答案: Express 的优点是线性逻辑：路由和中间件完美融合，通过中间件形式把业务逻辑细分，简化，一个请求进来经过一系列中间件处理后再响应给用户，再复杂的业务也是线性了，清晰明了。

### 8. 什么是中间件, 常用的第三方中间件有那些?

参考答案: Express是一个自身功能极简，完全是路由和中间件构成一个web开发框架：从本质上来说，一个Express应用就是在调用各种中间件。封装了一些或许复杂但肯定是通用的功能, 非内置的中间件需要通过安装后，require到文件就可以运行。

第三方中间件:

cookie-parser：解析cookie

ejs：ejs模板

express-session：解析session

http-errors：错误中间件处理

Moment：日期时间处理

Morgan：日志打印

Multer：文件上传

Nodemon：热更新

config-lite: 读取配置文件

### 9.express缺点是什么?

参考答案: Express 是基于 callback 来组合业务逻辑。Callback 有两大硬伤，一是不可组合，二是异常不可捕获。

### 10. 什么是同步, 什么是异步操作?

参考答案: 同步操作, 当代码运行到同步操作的代码时, 所在线程等待结果返回

而异步操作, 则是不会耽误代码继续执行, 当异步操作成功, 一般使用回调函数来处理异步成功的动作(常用在ajax/定时器/计时器等)

## 第7讲

### 1.ejs作用是什么?

EJS是一个Javascript模板库, 用来从JSON数据中生成HTML文件

2. koa的优点是什么?

首先，借助 promise 和 generator 的能力，丢掉了 callback，完美解决异步组合问题和异步异常捕获问题。  
其次，koa 把 express 中内置的 router、view 等功能都移除了，使得框架本身更轻量化。

3. koa缺点?

Connect/Express 的中间件基本不能重用，基本要重写

### 4. pug的作用是什么?

Pug是一款健壮、灵活、功能丰富的HTML模板引擎,专门为 Node.js 平台开发。Pug是由Jade 改名而来。是一种通过缩进（表示标签间的嵌套关系）的方式来编写代码的过程，在编译的过程中，不需要考虑标签是否闭合的问题。可以加快写代码速度，也为代码复用提供了便捷。

5. 什么是前后端分离项目 (JS渲染的项目又是什么)?

前端HTML页面通过AJAX调用后端的RESTFUL API接口并使用JSON数据进行交互

JS渲染的项目就是通过Ajax请求数据以后, 通过JS代码动态创建html的标签和数据等(一般右键查看网页源代码 是看不到渲染后的HTML标签的)

6. 什么是后端渲染项目?

浏览器请求静态网页资源, 服务器端会在后端把数据渲染到HTML页面上, 再把html文件内的字符串一起返回给浏览器进行展示的一种手段, Nodejs中体现为ejs和pug模板引擎

### 7. 设计一个前后端交互的流程?

前端编写HTML+CSS代码以后, 通过JS的Ajax进行网路请求, 请求后台指定的数据接口, 根据业务需求, 可以还需要传递参数等信息, 获取数据以后, 进行JSON解析和DOM创建展示.

服务器端 编写接口, 接收参数等, 连接数据库, 根据业务逻辑编写业务代码, 返回不同的数据

### 8. 分页功能, 前后端是如何交互的?

分页功能, 前后端都可以做, 如果是前端做分页的话, 根据翻的页码数, 截取后台返回的全部数据, 进行分段展示

如果是前后端配合做, 则需要传递当前页码数给后台, 后台会返回对应页码的数据

### 9. session和cookie的作用和区别?

session是区别于数据库存在的一种服务器临时存储技术, 它主要存储一些无需持久化的数据, 比如临时的登录状态信息等

cookie是存在于浏览器上的一种浏览器本地存储的方式, 同域名下的cookie不同标签页可以共享, 默认过期时间是浏览器关闭时, 而且在进行http请求时, 会自动带上浏览器全部的cookie发给后台, 后台也可以获取cookie, 设置可以在响应时, 想浏览器中设置cookie。

### 10. 跨域是什么, 如何解决跨域?

//端口，接口，域名不同则跨域

//[3、跨域解决办法](https://www.cnblogs.com/ysocean/p/9380551.html" \l "_label2)

* [①、response 添加 header](https://www.cnblogs.com/ysocean/p/9380551.html" \l "_label2_0)[②、JSONP 方式](https://www.cnblogs.com/ysocean/p/9380551.html" \l "_label2_1)[③、HttpClient 请求转发](https://www.cnblogs.com/ysocean/p/9380551.html" \l "_label2_2)[④、nginx 转发](https://www.cnblogs.com/ysocean/p/9380551.html" \l "_label2_3)

当ajax请求所在域名或接口和请求目标url的域名或接口, 有一个不同, 即发生了跨域请求, 浏览器会阻止这次ajax请求.

如果能让后台改代码, 可以让后台开启跨域支持/ 使用jsonp方式处理跨域

如果不能让后台改代码, 则需要在本地使用服务器, 进行代理转发

## 第8讲：

### 1、webpack的作用是什么，谈谈你对它的理解？

现在的前端网页功能丰富，特别是SPA（single page web application 单页应用）技术流行后，JavaScript的复杂度增加和需要一大堆依赖包，还需要解决SCSS，Less……新增样式的扩展写法的编译工作。所以现代化的前端已经完全依赖于WebPack的辅助了。

现在最流行的三个前端框架，可以说和webpack已经紧密相连，框架官方都推出了和自身框架依赖的webpack构建工具。

* React.js+WebPack
* Vue.js+WebPack
* AngluarJS+WebPack

### 2、WebPack工作原理

WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Sass，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。在3.0出现后，Webpack还肩负起了优化项目的责任。

这段话有三个重点：

* 打包：可以把多个Javascript文件打包成一个文件，减少服务器压力和下载带宽。
* 转换：把拓展语言转换成为普通的JavaScript，让浏览器顺利运行。
* 优化：前端变的越来越复杂后，性能也会遇到问题，而WebPack也开始肩负起了优化和提升性能的责任。

### 3、 webpack与gulp和grunt的区别是什么？

gulp强调的是前端开发的工作流程，我们可以通过配置一系列的task，定义task处理的事务（例如文件压缩合并、雪碧图、启动server、版本控制等），然后定义执行顺序，来让gulp执行这些task，从而构建项目的整个前端开发流程。

webpack是一个前端模块化方案，更侧重模块打包，我们可以把开发中的所有资源（图片、js文件、css文件等）都看成模块，通过loader（加载器）和plugins（插件）对资源进行处理，打包成符合生产环境部署的前端资源

## 第9讲：

### 1、为什么要进行模块化

目前前端的开发形势就是模块化和组件化；从软件工程学分析来说就是有了更好的可维护性、可复用性等好处；但是前端的主要语言js在es6之前却没有模块化功能，之前有使用require.js和sea.js但是推出es6的模块化之后，es6的模块化使用形式基本统一了。

### 模块化和组件化的区别

模块是把相似的代码功能封装在一个js当中，构成一个模块，一般改模块导出的形态是class或对象

组件化编程一般是指对页面上反复重复的dom 元素的封装

模块化常见于封装js 代码功能，组件化常见于封装DOM综合操作

组件是具体的：按照一些小功能的通用性和可复用性来抽象组件

组件化更多的关注UI部分，比如用户看到的弹出框，页脚，确认按钮等，这些组件可以组合成新的组件，又可以和其他组件组合组合成新的组件

模块是抽象的：按照项目业务划分的大模块

模块化侧重于数据的封装，一组相关的组件定义成一个模块，一个json对象可以是一个模块。

### 3、谈谈你对AMD和CMD的理解

AMD：

AMD推崇依赖前置，在定义模块的时候就要声明其依赖的模块

同样都是异步加载模块，AMD在加载模块完成后就会执行改模块，所有模块都加载执行完后会进入require的回调函数，执行主逻辑，这样的效果就是依赖模块的执行顺序和书写顺序不一定一致，看网络速度，哪个先下载下来，哪个先执行，但是主逻辑一定在所有依赖加载完成后才执行。

CMD：

CMD推崇就近依赖，只有在用到某个模块的时候再去require

CMD加载完某个依赖模块后并不执行，只是下载而已，在所有依赖模块加载完成后进入主逻辑，遇到require语句的时候才执行对应的模块，这样模块的执行顺序和书写顺序是完全一致的。

特点：AMD用户体验好，因为没有延迟，依赖模块提前执行了；CMD性能好，因为只有用户需要的时候才执行。

## 第10讲:

### 1、Mongodb的特点和使用场景：

实用性：MongoDB是一个面向文档的数据库，它并不是关系型数据库，直接存取BSON，这意味着MongoDB更加灵活，因为可以在文档中直接插入数组之类的复杂数据类型，并且文档的key和value不是固定的数据类型和大小，所以开发者在使用MongoDB时无须预定义关系型数据库中的”表”等数据库对象，设计数据库将变得非常方便，可以大大地提升开发进度。

可用性和负载均衡：MongoDB在高可用和读负载均衡上的实现非常简洁和友好，MongoDB自带了副本集的概念，通过设计适合自己业务的副本集和驱动程序，可以非常有效和方便地实现高可用，读负载均衡。

扩展性：在扩展性上，MongoDB有非常有效的，现成的解决方案。通过自带的Mongos集群，只需要在适当的时候继续添加Mongo分片，就可以实现程序段自动水平扩展和路由，一方面缓解单个节点的读写压力，另外一方面可有效地均衡磁盘容量的使用情况。

不适用的场景：在某些场景下，MongoDB作为一个非关系型数据库有其局限性。MongoDB不支持事务操作，所以需要用到事务的应用建议不用MongoDB，另外MongoDB目前不支持join操作，需要复杂查询的应用也不建议使用MongoDB。

### 2、什么是nosql，为什么使用nosql？

NoSQL，指的是非关系型的数据库。NoSQL有时也称作Not Only SQL的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。

NoSQL用于超大规模数据的存储。（例如谷歌或Facebook每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

今天我们可以通过第三方平台（如：Google,Facebook等）可以很容易的访问和抓取数据。用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了, NoSQL 数据库的发展却能很好的处理这些大的数据。

### 3、常见的关系型数据库和非关系型数据库都有哪些：

关系型数据库：Mysql，sqlserver，oracle等。

非关系型数据库：redis，mongodb等。

## 第11讲：

### 1、mongoose和mongodb的区别是什么？

mongoose是Node环境下异步操作数据的对象数据模型，仅限于Node环境下使用，是基于MongoDB的驱动。Mongoose是基于mongodb的框架类库，它方便和延伸了mongodb的使用，提出了重要的schema，和更方便的populate使用。

### 2、 简述mongoose中的schema和model区别？

Model是基于schema的实体表现，schema只负责集合结构的定义，model负责对集合进行CRUD操作

## 第12讲

### websocket是什么，有什么优点？

WebSocket是HTML5新增的协议，它的目的是在浏览器和服务器之间建立一个不受限的双向通信的通道，比如说，服务器可以在任意时刻发送消息给浏览器。

websocket的优点：

1. 持双向通信，实时性更强。
2. 减少通信量，只要建立起websocket连接，就一直保持连接，在此期间可以不断送消息，直到关闭请求。
3. 节约带宽。不停地轮询服务端数据这种方式，使用的是http协议，head信息很大，有效数据占比低， 而使用WebSocket方式，头信息很小，有效数据占比高

### 为什么要进行前后端分离？

1. 、提升开发效率，因为可以前后端并行开发，而不是像以前的强依赖
2. 、可以实现真正的前后端解耦（前端可以根据用户不同时期的体验需求迅速改版，后端对此毫无压力。同理，后端进行的业务逻辑升级，数据持久方案变更，只要不影响到接口，前端可以毫不知情。）、增加代码的维护性及易读性
3. 、前后端开发人员的职责更加专注：前端倾向于呈现，着重处理用户体验相关的问题；后端则倾向于业务逻辑、数据处理和持久化等。

## 第13讲

### koa与express的区别？该如何选择？



该如何选择？

如果你不想自己选择和搭建各种模块组件并有可能需要处理兼容问题，而喜欢一体的、快速集成开发，那么选择Express，Express是一套成熟的应用开发框架，而不仅仅提供核心模块。

如果是新项目，运行环境对ES7语法没有限制，对性能比较看重，团队有较高的定制化需求和技能，那么优先选择Koa；如果是Express的老项目，那么建议继续使用Express。

### 常见的web优化有哪些？

1. 、减少重复代码，压缩css，js代码大小
2. 、背景图片大小及数量
3. 、减少http请求，合理设置缓存
4. 、js中减少DOM操作，避免使用eval和 Function，减少作用域链查找
5. 、CDN加速
6. 、反向代理

### 域名与IP的关系？

在早期的互联网时代，没有那么多的主机，全部是用ip直接访问。可是IP不符合人类的记忆习惯，于是出现域名。

域名和IP的对应关系，在早期通过电脑的hosts文件直接解析，后来互联网的发展，越来越多的域名出现，单纯靠hosts解析显然是无法胜任的，于是出现解决方案就是沿用至今的DNS(Domain Name Server)，域名服务。

24. 如何进行网站性能优化

1. 从用户角度而言，优化能够让页面加载得更快、对用户的操作响应得更及时，能够给用户提供更为友好的体验。

2. 从服务商角度而言，优化能够减少页面请求数、或者减小请求所占带宽，能够节省可观的资源。

　　总之，恰当的优化不仅能够改善站点的用户体验并且能够节省相当的资源利用。

前端优化的途径有很多，按粒度大致可以分为两类，第一类是页面级别的优化，例如 HTTP请求数、脚本的无阻塞加载、内联脚本的位置优化等 ;第二类则是代码级别的优化，例如 Javascript中的DOM 操作优化、CSS选择符优化、图片优化以及 HTML结构优化等等。另外，本着提高投入产出比的目的，后文提到的各种优化策略大致按照投入产出比从大到小的顺序排列。

一、页面级优化

1. JavaScript 压缩和模块打包

2. 按需加载资源

3. 在使用 DOM 操作库时用上 array-ids

4. 缓存

5. 启用 HTTP/2

6. 应用性能分析

7. 使用负载均衡方案

8. 为了更快的启动时间考虑一下同构

9. 使用索引加速数据库查询

10. 使用更快的转译方案

11. 避免或最小化 JavaScript 和 CSS 的使用而阻塞渲染

12. 用于未来的一个建议：使用 service workers + 流

13. 图片编码优化

————————————————

版权声明：本文为CSDN博主「小胖梅」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/xm1037782843/article/details/80708533