年后知识点复习（2018年）

201712

ES6知识点梳理

http://es6.ruanyifeng.com/

let/const

```
let num=12;

const str='珠峰培训'; //=>const定义变量的值不可以被修改 （redux中定义的行为标识一般都使用const，这些行为从代码角度被规定为不可修改...）

//=>汇总定义变量的方式
var / function / let / const / class（通过class创建出来的类只能被new不能被当做普通函数执行）/ import
```

let 和 var 的对比（重点：变量提升、闭包...）

```
//=>1、let不存在变量提升，var存在变量提升
console.log(num); //=>Uncaught ReferenceError: num is not defined
let num=12;

以后使用ES6中的let最为定义变量的主要语法，我们需要把后期使用的变量提前声明或者定义
let num,
    str='珠峰培训',
    obj=null,
    oo=null;
...
oo={};

//=>2、let会产生块级作用域
闭包：函数执行产生一个不销毁的私有作用域，这个作用域存在两个作用 [ 保护 / 保存 ]

//=>单例模式
let loginRender=(function(){
    let num=12;

    let fn=function...
```

```
    return {
        fn  (=>fn:fn)
    }
})();

//=>高阶函数（REACT中的高阶组件也是这个原理）
Function.protptype.bind = function bind(context){
    //->this:fn
    let outerAry=Array.prototype.slice.call(arguments,1);
    let _this=this;
    return function(){
        let innerAry=[].slice.call(arguments);
        _this.apply(context,outerAry.concat(innerAry));
    }
}

setTimeout(fn.bind(context,param1....),1000);
document.onclick=fn.bind(context,...);
```

解构赋值

最常用的就是对数组和对象的解构赋值；在这里会引发出...的应用；

```
//=>模块导入
import {createStore,combineReducer...} from 'redux';

//=>给对象重新赋值
let obj={name:'珠峰培训',age:9};
obj={...obj,name:'哈哈哈'}; //=>利用...是展开运算符来完成的

//=>解析服务期端返回的数据
let data={code:0,num:12,ary:[12,23,34]};
或者num属性的值和，数组最后一项的值
let {num:reNum,ary:[,,last]}=data;
//reNum:12
//last:34

//=>组件封装，给传递进来的对象参数值做一些默认值处理
function ajax({
    url=null,
    dataType='text',
    ...
}={}){
    ...
}
```

```
23.
24. ajax({
25.     url:xxx,
26.     data:null,
27.     dataType:'json',
28.     async:true,
29.     cache:false,
30.     success:function...,
31.     ...
32. });
33.
34. //=>实现变量值交换
35. let a=12;
36. let b=13;
37. [b,a]=[a,b];
38.
39. 再处理具体需求（例如：数据绑定的时候），我们会把某一个数组或者对象中的某一部分获取到，然后进行绑定...
40. class Login extends React.Component{
41.     render(){
42.         let {name,age}=this.props;
43.         return <div>
44.         </div>;
45.     }
46. }
```

```
<Login name='xxx' age=8/>

...
```

...的作用

```
1.  //=>展开运算符
2.  obj={...obj,name:xxx}
3.  Math.max(...ary) <=> fn(...ary)
4.
5.  //=>拓展运算符
6.  let [a,...c]=ary; //=>a是数组第一项
    c是把数组剩下的项获取到，成为一个新的数
    组
7.  let [...copyAry]=ary; //=>浅克隆
8.  let copyAry=JSON.parse(JSON.strin
    gify(ary)); //=>深克隆
9.
10. //=>剩余运算符
11. function fn(a,...arg){
12.     a:12
13.     arg:[23,34,45,56]
14.     箭头函数中没有arguments，所以我们
    需要使用...arg获取传递进来的实参值，以
    此代替arguments（...arg获取的是一个数
    组，arguments是一个类数组）
15. }
16. fn(12,23,34,45,56);
```

箭头函数

箭头函数中没有明确的this指向，它的this是继承其上下文(和他爹有关系)

```
/*
Function.prototype.bind=(context,...arg)=>{
    //=>this:window
    //=>箭头函数不能乱用，会导致this出现问题，一般外层函数很少使用箭头函数处理
};
*/

Function.prototype.bind=function bind(context,...outerArg){
    return (...innerArg)=>{
        //=>this:和外层函数一样了,也就是需要处理的函数fn
        this.apply(context,outerArg.concat(innerArg));
    };
}
setTimeout(fn.bind(obj,10,20),1000);
```

JS中的this

```
1、函数执行，看其是否有执行主体（就看函数执行前面是否有点），没有点，函数中的this是window（在严格模式下是undefined），有点，点前面是谁THIS就是谁

2、事件绑定中，一般情况下给当前元素的某个事件行为绑定方法，方法中的this是当前元素本身（在DOM2事件绑定，并且运行在IE6~8低版本浏览器中 [attchEvent]，方法中的this是window）

3、使用call/apply/bind可以强制改变this的指向

4、类主体中的this一般都是当前类的一个实例

5、箭头函数中没有自己的this
```

class

JS中的OOP（面向对象）：对象、类、实例

我们涉及的插件、组件、类库、框架等都是基于类来完成的，例如：Vue、React、ReactDOM、JQ、ZEPTO、Swiper、Iscroll、Bootstrap…都是类

prototype、__proto__

Array.prototype.distinck=function…

```
class Person {
    constructor(num1, num2) {
        console.log(num1, num2);
        this.xxx = 100;//=>实例私有的属性
    }

    //=>以下方法是写在Person.prototype上的(实例公有的属性和方法)
    say() {
        console.log(`i can say`);
    }

    //=>把当前类当做普通对象，设置一些跟类有关的属性和方法（和实例没有任何的关系）
    //=>Person.sum=function(){}
    static sum() {

    }
}
Person.privateProp = 100;
Person.prototype.publicProp = 200;
```

类的继承

```
class Fn {
    constructor(a, b) {
        //=>a:100 b:200  this:p
        this.x = 100;
    }

    getX() {
        console.log(this.x);
    }
}

class Person extends Fn {
    constructor(...arg) {
        //=>this:p
        super(...arg);//=>super(100,200)
        this.y = 200;
    }

    getY() {
        console.log(this.y);
    }
}

let p = new Person(100, 200);
console.log(p.x, p.y);//=>100 200
```

其它的继承方式

- 原型
- CALL继承
- 寄生组合式继承（和ES6中的继承实现效果一样）

你是否编写过插件？你都用过哪些插件？

- swiper实现H5滑屏的
- iscroll实现局部滚动的
- Ecarts实现图表的（Highcharts、d3.js、svg.js…）
- jquery dialog 弹出层
- jquery validate 表单验证
- jquery datepicker 日历插件
- jquery Draggable 拖拽
- bootstarp
- …

编写插件？
1、基于面向对象封装
2、通用原则（例如：实现一个dialog插件，需要我们尽可能的支持更多种弹出效果以及动画效果等）

```
提示内容都是用户自己设定的

类型：
警告提示/确认提示/需要输入的(登录框)...

运动：
匀速/非匀速/是否支持拖拽...

是否能够自定义样式
...
```

3、容易使用（完善的功能需要用户自己进行config配置，我们需要把配置项默认以及依赖规划好，让开发易用）
4、利于升级更新（新版本诞生后，老版本是否可以继续使用，如果升级是否麻烦等

技巧：升级新的版本，老的配置参数不变，使用一些新的处理字段，逐步实现老和新都兼容的模式，最后统计使用每个版本使用量，当老版本过渡完成后，去掉老的配置字段等）

同步异步编程

promise

async / await

JS是单线程（一次只能处理一件事情），异步是在这个基础上，增加等待处理的机制

> JS中有主任务队列和等待任务队列，主任务队列中的事情按顺序依次执行，遇到异步任务，先存放在等待任务队列中，当主任务队列完成后，在去等待任务队列中，把相关的事情拿到主任务队列中执行
>
> - 定时器（node => setImmediate / process.nextTick）
> - 事件绑定
> - ajax中存在异步编程
> - 回调函数也可以理解为异步编程
> - 基于Promise可以对异步操作进行处理（面试回答ES6的重点）
> - 基于async / await 来处理异步操作
> - …

1. setTimeout/setInterval：设置定时器，会把当前任务（过多久执行回调函数的任务）放在等待队列中，当主任务队列完成后执行
2.
3. process.nextTick：也是异步的，但是它放在主任务队列的最尾部，也就是说它肯定会先于异步任务之前完成
4.
5. setImmediate：异步操作，放在所有任务（包含异步任务）的最后完成

JS中的设计模式

- Promise设计模式
- 发布订阅模式
- MVVM（MVC）框架设计模式：基于数据驱动视图渲染，实现双向数据绑定
- 单例模式（闭包）
- 构造原型模式（OOP）
- …

```
//=>发布订阅原理：JQ
let $plan=$.Callbacks(); //=>创建一个事件池（容器）

$plan.add(function...); //=>向容器中增加需要处理的任务
$plan.remove(function...); //=>从容器中移除一些不需要处理的任务
...

$plan.fire([parameter]...); //=>遍历容器中的每一个任务，并且依次执行 [parameter] 给每一个任务传递的实参

----------------------
1、backbone实现mvc的原理，主体就是基于发布订阅完成的
2、redux中的subscribe就是基于发布订阅完成的，当dispatch任务派发成功，会把基于subscribe增加的方法依次执行
3、在一些具体的业务逻辑中，我们也会使用发布订阅模式，来控制当某个任务完成（或者某个条件触发）后，依次执行N多个其它的任务
4、DOM2中attachEvent的兼容处理，我们也是自己基于发布订阅模式来构建一个虚拟事件池解决的
```

```
...

```

```
//=>MVVM的原理（vue2.0）
let data = {
    msg: 'hello word'
};
Object.defineProperty(data, 'msg', {
    get() {
        console.log(1);
        return 'ok';
    },
    set(val) {
        console.log(2);
    }
});
console.log(data.msg);
data.msg = 100;

//=>proxy
据说vue3.0版本将基于ES6中的proxy实现（谣言不可信）
```

其它需要自己了解的

Set / Map ：做一下了解即可

```
1. //=>快速实现数组去重
2. var set1 = Array.from(new Se
   t([1,1,2,2,33,'33',44,'44'
3. ]));
```

Interator（先看Generator）：不怎么用，但是大家把 for of 看一下

```
1. for of 能遍历数组，但是不能直接遍历对
   象，对象属于不可被迭代的
```

Symbol：创建一个唯一值

ES6中的模板字符串

ES6中不管是数组，还是对象，或者字符串，都新增加一些常用的方法

…

DOM

项目源码（html/css/js/img...）存放在服务器上的 -> 浏览器地址栏输入网址（或者IP地址）

```
1、到DNS服务器上进行域名解析（=>获取到服务器外网IP地址）
2、通过外网IP找到服务器
3、通过端口号找到相关的服务
4、服务器的相关服务根据请求的文件地址，获取到文件中的源代码（文件读取 / 数据读取）
5、服务器端把准备好的数据返回给客户端

----

客户端和服务器端信息的交互，是基于传输协议完成的（http/https/ftp...）
请求+响应：HTTP事务
传输的内容：HTTP报文（起始行、首部[头]、主体...）

TCP的三次握手和四次挥手
```

-> 浏览器获取内容后开始进行解析（内核/渲染引擎）

```
1、浏览器获取到HTML页面源代码后会解析为DOM树（HTML结构） 和 渲染树（依据样式等从上到下进行渲染，最后绘制成图形和页面）

https://www.cnblogs.com/ranyonsue/p/8328120.html
关于DOM的重排重绘（JS中性能优化主要一点之一）

现代的浏览器都有渲染队列的机制（如果连续修改元素的样式，不会立即触发重排，而是把需要修改的内容先存放在一个队列中，当发现没有要修改的了，浏览器统一把队列中的样式进行修改，引发一次重排）

建议：把修改样式的操作尽可能写在一起或者合并
div.style.xxx=xxx;
div.style.xxx=xxx;
或者
div.style.cssText='xxx:xxx,xxx:xxx';

都写完后，我们在统一获取自己需要的最新样式
console.log(div.offsetLeft);
```

```
16.
17.  或者建议：基于createDocumentFragment把需要修改样式的元素存放在文档碎片中，把样式修改完成后，在把元素重新放到页面中
```

DOM事件模型

- DOM0事件和DOM2事件对比
- 事件对象 e (MouseEvent/KeybordEvent)
- 事件代理（委托）：事件传播
- 移动端事件

```
1.  tocustart\touchmove\touchend ...
2.
3.  1、移动端click事件300MS延迟问题？
4.  2、zepto中tap处理点击出现的“穿透”问题？
5.  3、移动端不怎么支持keydown/keyup，使用input事件来处理
6.  ...
```

DOM数据渲染

1. 1、VUE的 MVVM 双向数据绑定原理？
2. 2、REACT中JSX虚拟DOM到真实DOM转换的原理？
3. 3、VUE或者REACT中的差异渲染（和之前一样的不重新渲染，只渲染改变的部分）

AJAX和跨域

- ajax
  - HTTP状态码
  - 步骤
  - 基于promise封装ajax库
  - fetch
  - axious：相当于传统ajax的优缺点
  - …

- 跨域
  - jsonp：利用script标签不存在域的概念（弊端：都是GET请求）
  - cors：跨域资源共享（一般是服务端设置可以多源访问，客户端使用的还是ajax）
  - 其它跨域方式：document.domain / postMessage / window.name+iframe…
  - webpack配置代理

    - …

框架

- 生命周期
- 路由：HASH路由
- 组件：高阶组件
- vuex / redux
- axious
- vue
    - 指令
- react
    - 属性状态

最主要的是突出自己可以使用框架，并且可以基于框架完成项目（最好是独立完成）

webpack && git

回去后自己搭建一个webpack

node（优势不是必须）

写伪API

- npm包管理

- node基础API
- 经常使用的第三方模块
- Express
- socket.io
- mongdb
- …