第三阶段知识复盘-JavaScript高级

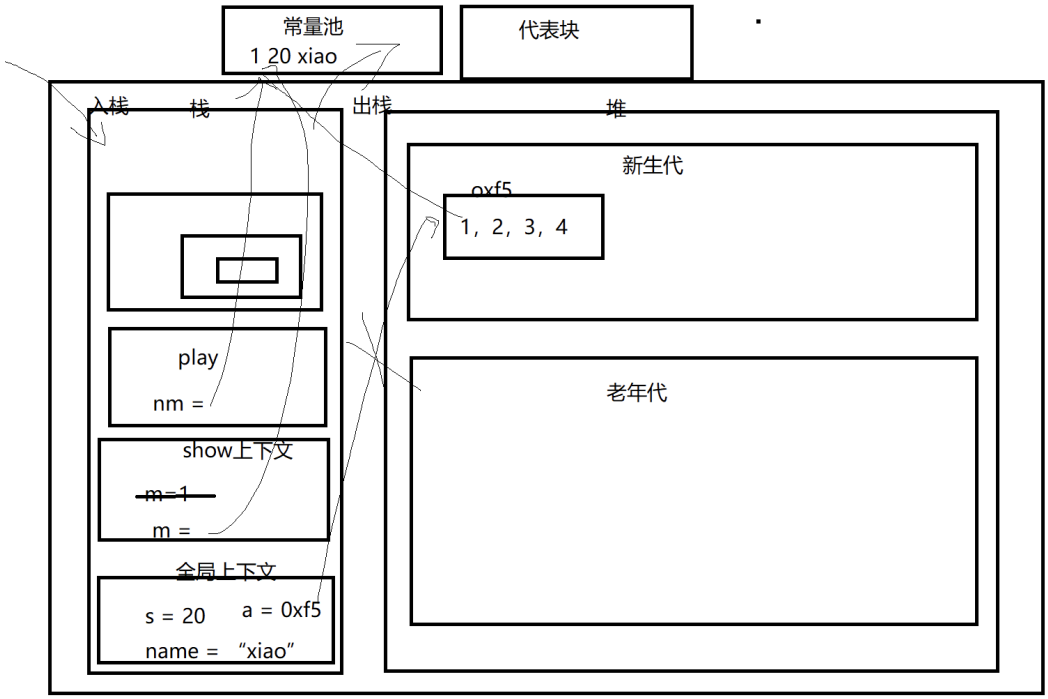
一、内存分析

变量的定义：内存里面表现出来

数据类型：也是内存里面指定空间大小

程序：一定是输入、一定会输出

绘制内存图



说明：

1. 栈空间比堆空间小很多，栈主要存放基本数据类型变量
2. 堆空间存放引用类型数据
3. 栈满足先进后出，程序一开始执行默认产生一个全局上下文对象（全局空间）
4. 栈里面还有函数上下文对象，调用一个函数的时候，产生这个空间，函数执行完毕后弹出栈
5. 递归调用、每次递归都会产生一个函数上下文空间。一层一层开始执行，一层一层弹出
6. 堆空间会分为新生代和老年代
7. 新生代表达刚开始创建对象存放空间，这个空间垃圾回收机制重点扫描的
8. 老年代空间，当新生代空间里面对象，经过多次扫描回收都还存在，自动转入老年代空间。
9. 内存空间常量池：主要存放我们常见的字面量（1，2，xiaowang）
10. 代码区：存放函数或者类等等

深克隆浅克隆

栈溢出错误：

```
//Outputs: Uncaught RangeError: Maximum call stack size exceeded
```

二、作用域

变量的分类：

1. 局部变量（函数变量）：这个变量作用范围函数内部，局部
2. 全局变量：全局的变量，放在全局上下文对象中

```
var s = 100
function show(){
    var m = 20
}
```

研究局部变量和全局变量特点

1. 局部变量放在函数中使用，只能在这个函数内部生效
2. 全局变量放在全局定义，可以在任何地方使用

作用域

当前变量的作用范围，

1. 函数作用域：程序要使用某个变量，首先从自己函数中寻找
2. 全局作用域：在全局环境下使用变量，默认从全局找

作用域链：

如果要使用某个变量，首先从自己自身环境中寻找。在往上层作用域寻找，找全局

```
var age = 20
function show(){
    var age = 10
    function play(){
        console.log(age)
    }
}
```

函数中this指向

块作用域：let const才形成的。一般来说，只有涉及研究let和const特点的时候，我们提到块作用域概念

```
{

}
```

三、变量关键字区别

let const var

扩展内容：

1. let const其实也有变量提升。let const提供暂时性死区。导致就算变量提升，也无法使用

   ```
   console.log(i) //
   let i = 10
   ```

2. 如果一个变量没有使用let var const，这个变量是一个全局变量

```
function show(){
    n = 44
}
show()
console.log(n);
console.log(this);
```

实战：

```
let array = []
for (var index = 0; index < 5; index++) {
    array.push(function(){
        console.log(index)
    })
}
array[0]()
array[1]()
array[2]()
```

四、递归的练习题

简答的累加

数列求和：1、4、7、10、13

```
function sum(n){
    //一定找到递归出口
    if(n==1){
       return 1
    }
    return sum(n-1)+3
}

sum(10)
```

求1-100的和

```
function show(n){
    if(n==1){
       return 1
    }
    return show(n-1)+n
}
show(100)

//递归的第一次：show(99)+100
//递归的第二次：show(98)+show(99)+100....show(1)
```

特点：

1. 首先要在函数中执行，自己调用自己
2. 递归一定要出口，死递归栈溢出
3. 分析每次递归的结果，这个结果可能会作为下次的一些条件

五、IIFE立即执行函数

在ES6没有出来之前，我们前端没有官方模块化的

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="index01.js"></script>
    <script src="index02.js"></script>
</head>
<body>

</body>
</html>
```

模块化要解决的就是JS代码的问题

```
(function(){
    // index01.js
    var m = 10
})()
```

在外部js代码中引入IIFE来定义一个局部函数作用域，每个函数中变量相互不冲突

有了官方模块化后，这些问题就已经不考虑了。

你们写的ES6代码，通过webpack打包为ES5代码，打包后代码默认会用IIFE的这种方式来隔离的作用域

六、对象拷贝

内存分配

浅拷贝：对象拷贝的时候，外层肯定要拷贝，内层如果引用类型，不会生成新的对象，直接拷贝地址

深拷贝：不管对象有多少层，我们拷贝的时候，每一层都会生成一个新的对象

方案：

浅拷贝：扩展运算符、forin循环、Object.assign()

深拷贝：JSON.stringify、jQuery提供extend、第三方load提供_cloneDeep() 、手写递归

七、函数防抖节流

跟时间绑定有关系

防抖：触发事件后，在n秒内默认只执行一次，触发事件n秒内又触发这个事件，会重新计算时间。

```
function debounce(fn,wait){
    var timer = null;
    return function(){
        if(timer !== null){
            clearTimeout(timer);
        }
```

```
        timer = setTimeout(fn,wait);
    }
}

function handle(){
    console.log(Math.random());
}

window.addEventListener("resize",debounce(handle,1000));
```

节流：连续触发事件，但是在n秒内默认只会执行一次。

```
function throttle(func, wait) {
    let timeout;
    return function() {
        let context = this;
        let args = arguments;
        if (!timeout) {
            timeout = setTimeout(() => {
                timeout = null;
                func.apply(context, args)
            }, wait)
        }

    }
}
function handle(){
    console.log(Math.random());
}

window.addEventListener("resize",throttle(handle,1000));
```

八、开发方式

前端开发：

1. 面试对象开发

   比如你们用到数组、日期、Object都是对象

   ```
   获取对象属性
   obj.id
   obj["id"]
   修改对象属性
   obj.id = 2
   查询对象
   for(const params in obj){

   }
   ```

   操作的对象大部分都是原生对象，很少自己封装对象

2. 函数式编程

   前端开发用的很多的模式

   项目中设计一个工具utils.js

   ```

   ```

```
//日期格式化
 export const DateFormat = ()=>{
     //队列逻辑
 }

//本地存储
export const storageHandler = ()=>{

}
//电话号码过滤

export const phoneFilter = ()=>{

}
```

在Vue中使用

```
import {DateFormat} from "./utils.js"

methods:{
    changeDate(){
        const res = DateFormat("yyyy-MM-dd")
    }
}
```

面向对象来设计工具

utils2.js

```
class Utils{
    //第一个工具
    DateFormat(){

    }
    //
    storageHandler(){

    }
}

export default Utils
```

Vue中使用

```
import Util from "./utils2.js"

methods:{
    changeDate(){
        const u = new Util()
        const res = u.DateFormat("yyyy-MM-dd")
    }
}
```

utils3.js封装工具

```
class Utils{
    //第一个工具
    static DateFormat(){

    }
    //
    static storageHandler(){

    }
}

export default Utils
```

Vue中使用

```
import Util from "./utils3.js"

methods:{
    changeDate(){
        const res = Util.DateFormat("yyyy-MM-dd")
    }
}
```

类似于你们用官方工具Math

```
methods:{
    changeDate(){
        const res = Math.random()
    }
}
```

开发过程中，前端面向对象和函数式编程都有

```
user?.name?.play
user??name??m
```