准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->JSCORE->day07、day08

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->JSCORE->day07、day08

如果因为某些原因错过直播讲解，不用着急！直接看小程序中视频的回看即可。

如果tmooc不能用了，立刻看小程序视频，小程序视频比tmooc要全，甚至有扩展的新知识。

如果学有余力，将来有能力拿更高薪资，小程序中所有视频必须都要看。但是不必现在都看完。只要在笔试面试前都看完即可！

问题: 个人版小程序有功能限制：禁止在线播放视频

解决: 长按视频连接地址，完整复制视频连接地址和验证码到浏览器打开。通过百度网盘下载后观看。

3. 今日课程中包含的英语单词的翻译：小程序->单词->JSCORE->day07、day08

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例/视频，至少写三遍: (如果需要画图，必须亲手画图)

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩14天）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

谁问的问题越多！谁的培训费交的越值！

正课: ES6 —— 难

1. 参数增强

2. 解构

3. class

4. Promise

一. 参数增强:

对应小程序视频: 小程序->在线->JSCORE->day07 5. 参数增强 ...

1. 参数默认值

(1). 什么是: 定义函数时，希望将来就算没有给实参值，形参变量也有一个默认值可以使用。

(2). 如何: function 函数名(形参变量=默认值){ ... }

(3). 原理:

a. 如果调用函数时只要传了实参值（!==undefined），都优先使用调用者传入的自定义实参值

b. 只有调用函数时没有传入实参值时，才被迫使用=后的默认值作为参数变量的默认值使用。

(4). 示例: 为函数形参变量指定参数默认值

1\_default.html

|  |
| --- |
| <script>  //定义一个函数可以输出我的自我介绍  // 默认值  function intr(intrSelf="主人很懒，什么也没留下"){  document.write(`<h1>我的自我介绍: ${intrSelf}</h1>`)  }  //调用者传入了自我介绍的内容  intr("you can you up, no can no bibi");  //调用者没有传入自我介绍的内容，也想有默认值占位  intr();  //如果恶意传入""  intr(""); //因为""不等于undefiend，所以不启用默认值！而使用""  //如果恶意传入undefined  intr(undefined);//因为传入的值也是undefined，所以认为什么都没传入，所以启用默认值！  </script>  运行结果:  我的自我介绍: you can you up, no can no bibi  我的自我介绍: 主人很懒，什么也没留下  我的自我介绍:  我的自我介绍: 主人很懒，什么也没留下 |

2. 剩余参数：

(1). 什么是: 专门用于代替arguments接收不确定个数的实参值的新语法

(2). 为什么: arguments的缺点:

a. 箭头函数中不能使用arguments

b. 不是纯正的数组，无法使用数组家的函数

c. 只能接收所有实参值，无法有选择的接收部分实参值

d. 名字太长，且没有意义！

(3). 何时: 今后只要在ES6中获取不确定个数的实参值，都用剩余参数语法代替arguments

(4). 如何: 定义函数时

function 函数名(形参变量, 形参变量, ...数组名){ }

(5). 结果: ...运算符专门收集除前几个形参之外剩余的实参值，放入一个自定义数组中返回.

(6). 示例: 使用剩余参数语法计算不确定个数的数字的和

2\_rest.html

|  |
| --- |
| <script>  //错误:  // var add=()=>{  // var sum=0;  // for(var elem of arguments){//报错: arguments is not defined  // sum+=elem  // }  // return sum;  // }  //正确:  // var add=(...arr)=>{  // //arr:[1,2,3]  // var sum=0;  // for(var value of arr){  // sum+=value  // }  // return sum;  // }  //正确: 证明arr是一个纯正的数组  // var add=(...arr)=>{  // 数组家的  // var result=arr.reduce(function(捐款箱,elem){  // return 捐款箱+elem  // },0);  // return result;  // }  //正确: 终极简写，时髦！—— 学有余力的同学  var add=(...arr)=>arr.reduce((捐款箱,elem)=>捐款箱+elem,0);  console.log(add(1,2,3));//6  console.log(add(1,2,3,4,5));//15  运行结果:  6  15 |

(7). 示例: 使用剩余参数计算薪资

3\_rest2.html

|  |
| --- |
| <script>  //定义一个计算薪资的函数  // 收集除第一个ename之外剩余的其它实参值  function jisuan(ename,...arr){  //因为arr是一个纯正的数组  var total=arr.reduce(function(捐款箱,elem){  return 捐款箱+elem  },0);  console.log(`${ename}的总工资是：${total}`)  }  jisuan("Li Lei",10000,2000);  jisuan("Han Meimei",3000,4000,5000,1000)  </script>  运行结果:  Li Lei的总工资是：12000  Han Meimei的总工资是：13000 |

(8).优点:

a. 箭头函数支持

b. 是一个纯正的数组，可使用数组家函数

c. 可有选择的获取部分实参值，不一定非要获取全部实参值

d. 可自定义数组名，简单，见名知意

3. 打散数组和对象

(1). 什么是: 将数组或对象打散为多个实参值，再传入函数中

(2). 何时: 如果要调用的函数需要多个实参值，但是多个实参值却是放在一个数组中给的，出现不一致，就可先打散，再传参

(3). 如何: 调用函数时:

要调用的函数(...数组名)

(4). 原理: 先打散数组为多个值，再分别传给要调用的函数的形参变量

(5). 总结:

a. 定义函数时，形参列表中的...表示收集剩余参数

b. 调用函数时，实参值中的...表示打散数组再传参

(6). 示例: 查找数组中的最大值和最小值

4\_spread.html

|  |
| --- |
| <script>  //回顾: Math.max() Math.min()  console.log(`最大值: ${Math.max(2,5,7,1)}`);  console.log(`最小值: ${Math.min(2,5,7,1)}`);  //通常程序中保存多个值都是放在数组中保存  var arr=[2,5,7,1];  //想获得数组中多个元素值中的最大值和最小值  //错误:  console.log(`最大值: ${Math.max(arr)}`);//NaN  console.log(`最小值: ${Math.min(arr)}`);//NaN  //说明Math.max()和Math.min()不支持数组中查找最大值和最小值  //解决: 先打散数组再传参  //正确:  console.log(`最大值: ${Math.max(...arr)}`);//7  console.log(`最小值: ${Math.min(...arr)}`);//1  </script>  运行结果:  最大值: 7  最小值: 1  最大值: NaN  最小值: NaN  最大值: 7  最小值: 1 |

(7). 时髦: 其实都是先打散再粘起来

a. 复制一个数组: var arr2=[...arr1];

b. 复制一个对象: var obj2={...obj1};

c. 合并两个数组为一个新数组: var arr2=[...arr1,...arr2];

d. 合并两个对象为一个新对象: var obj3={...obj1, ...obj2};

(8). 示例: 使用...简写复制数组和对象，以及合并数组和对象

5\_spread2.html

|  |
| --- |
| <script>  //复制一个数组  var arr1=[1,2,3];  var arr2=[...arr1];  console.log(arr2);  console.log(arr1==arr2);//false  //复制一个对象  var lilei={  sname:"Li Lei",  sage:11  }  var lilei2={...lilei};  console.log(lilei2)  console.log(lilei==lilei2);//false    //合并两个数组  var arr1=[1,2,3];  var arr2=[6,7,8];  var arr3=[...arr1,4,5,...arr2];  console.log(arr3);  //合并两个对象:  var lilei={  sname:"Li Lei",  sage:11  };  var lilei2={  className:"初一2班",  intr:function(){  console.log(`I'm Li Lei, I'm 11`)  }  };  var lilei3={  ...lilei,  phone:"iPhoneX",  ...lilei2  }  console.log(lilei3);  </script>  运行结果:  [1, 2, 3]  false  {sname: "Li Lei", sage: 11}  false  [1, 2, 3, 4, 5, 6, 7, 8]  {sname: "Li Lei", sage: 11, phone: "iPhoneX", className: "初一2班", intr: ƒ} |

二. 解构:

1. 什么是: 从一个大的数组或对象中只提取出个别想用的成员保存在变量中，单独使用

2. 为什么: 在每次使用数组或对象成员时，总要arr[i]或对象.属性名——麻烦

3. 何时: 今后只要希望从一个大的数组或对象中只提取出个别成员单独使用时，就可用解构

4. 如何: 3种情况:

(1). 数组解构:

对应小程序视频: 小程序->在线->JSCORE->day07 6. 解构(上) ...

a. var [变量1, 变量2,...]=数组

0 1

b. 结果: =右边数组中对应位置的元素值会自动赋值给=左边[]中相同位置的变量

变量1=数组[0]

变量2=数组[1]

c. 强调: =左边的[]不是创建数组的意思！仅仅是将=左边的变量们装扮成数组的样子，骗取=右侧数组的信任，以便获得=右侧数组中对应位置的值。

d. 固定套路: 不声明第三个变量，交换两个变量的值: [b,a]=[a,b]

e. 示例: 使用解构提取数组中保存的年月日，以及交换两个变量的值

6\_destruct\_arr.html

|  |
| --- |
| <script>  var arr=[2020,6,25];  // 年 月 日  //var [y,m,d]=arr;  //console.log(`今年端午节放假:${y}年${m}月${d}日`)  //错误:  //var [m,d]=arr; //m=arr[0]=2020, d=arr[1]=6  //正确:  var [ ,m,d]=arr; //m=arr[1]=6, d=arr[2]=25  // 0 1 2  console.log(`今年端午节放假:${m}月${d}日`)  //不能声明第三个变量，交换两个变量的值  var a=3;  var b=5;  // 先将a和b放入一个数组中  [b,a]=[a,b]  //再用b和a解构出来对应位置的元素值  console.log(a, b);//5 3  </script>  运行结果:  今年端午节放假:6月25日  5 3 |

(2). 对象解构

对应小程序视频: 小程序->在线->JSCORE->day07 6. 解构(上) ...

a. 什么是: 提取出对象中个别属性值或方法，保存到单独变量中简化使用

b. 如何: var {属性名1: 变量1, 属性名2:变量2, ...:... }=对象

c. 结果: =左边对应属性名后的变量会自动收到=右边对象中同名属性的属性值

变量1=对象.属性名1

变量2=对象.属性名2

d. 其中:

1). {}不是创建对象的意思，仅仅是将等号左边的多个变量装扮成对象的样子，来骗取等号右边的对象的信任，从而自动提取出相同属性名的属性值保存到对应变量中

2). =左边{}里的属性名专门用于配对

e. 问题: 如果被解构出来的方法中用到this，this就是会变成window！

所以: 如果对象方法中用到this，不应该用这种方式解构！

f. 简写: 多数情况，对象中的成员名已经很简洁，很见名知意了，不需要改名。但是我们又不想把相同的属性名和变量名写两遍，其实是需要写一遍成员名即可:

1). var {成员名1, 成员名2,...}=对象

2). 相当于: var {成员名1:同名变量1, 成员名2:同名变量2,...}=对象

3). 结果: 一个名字两用: 既当做配对时的属性名，又当做将来单独使用时的变量名。

g. 总结: 今后对象结构时，=左边的{}中都使用简写，只写一个成员即可！

h. 示例: 从对象中解构出个别属性和方法单独使用

7\_destruct\_obj.html

|  |
| --- |
| <script>  var user={  uname:"dingding",  phone:"18301092802",  sex:1,  age:20,  intr:function(){  console.log(`I'm ${this.uname}, I'm ${this.age}`);  },  login:function(){  console.log("登录...")  },  logout:function(){  console.log("注销...")  },  changePwd:function(){  console.log("修改密码...")  }  }  //在当前页面中只想使用uname和changePwd两个成员  // 属性名:变量名, 属性名:变量名  // var {uname:u, changePwd:change}=user;  // //u=user.uname  // //change=user.changePwd  // console.log(`欢迎 ${u}`);  // change();  //如果我们觉得对象中的属性名已经很简洁，且见名知意了，不想改名  //var {uname:uname, changePwd:changePwd}=user;  //简写:一个名字两用,既当做配对的属性名，又当做将来单独使用时的变量名  var {uname, changePwd, intr}=user; //今后对象解构基本都只写一个名字  console.log(`欢迎 ${uname}`);//全局变量  changePwd();  //用旧方法调用  user.intr();//this->user this.uname=user.uname=dingding  //解构后再调用  intr();//this->window this.uname=window.uname=dingding  //this.age=window.age=undefined  </script>  运行结果:  欢迎 dingding  修改密码...  I'm dingding, I'm 20  I'm dingding, I'm undefined |

(3). 参数解构

对应小程序视频: 小程序->在线->JSCORE->day08 1. 解构(下) ...

a. 问题: 如果一个函数多个实参值都不确定有没有，但是又要求传入的实参值必须固定传给指定的形参变量才行

b. 错误解决: 单靠默认值无法解决所有需求！

c. 正确解决: 参数解构: 2步:

1). 定义函数时，先将形参列表装扮为一个对象的样子

function 函数名({

配对儿

属性名1: 形参1,

属性名2: 形参2,

属性名3: 形参3

}){

}

2). 调用函数时，所有实参值也必须放入一个相同的对象结构中传入

函数名({

属性名1:实参值1,

属性名2:实参值2,

属性名3:实参值3

})

d.结果: 形参1=实参值1, 形参2=实参值2, 形参3=实参值3

e. 简写: 定义函数时，通常用于配对的属性名和形参变量名几乎都是相同的，所以，形参列表对象中的名称只要写一个，就可以一个名字两用。而且，一旦改为一个名字，就可以继续支持参数默认值语法了。

function 函数名({

//一个名字两用: 既用作将来配对的属性名，又用作接收实参值的形参变量名

形参名1=默认值,

形参名2=默认值,

... ...

})

f. 示例: 定义一个点套餐的函数，支持各种需求的客户随意更换套餐中的内容

8\_destruct\_arguments.html

|  |
| --- |
| <script>  //要定义一个点套餐的函数  //一个套餐中包含三种商品:  //主食: zhushi 默认: 香辣鸡腿堡  //小吃: xiaochi 默认: 薯条  //饮料: yinliao 默认: 可乐  // function order({  // //配对儿 接收实参值  // //属性名 : 形参  // zhushi:zhushi, //不支持默认值  // xiaochi:xiaochi, //不支持默认值  // yinliao:yinliao //不支持默认值  // }){  function order({  //一个名字两用: 既当做配对的属性名，又当做接收实参值的形参变量名  zhushi="香辣鸡腿堡",  xiaochi="薯条",  yinliao="可乐",  }){  console.log(`  您点的套餐是:  主食:${zhushi},  小吃:${xiaochi},  饮料:${yinliao}  `)  }  //第一个用户: 香辣鸡腿堡， 薯条， 可乐  order({  zhushi:"香辣鸡腿堡",  xiaochi:"薯条",  yinliao:"可乐"  })  //第二个用户着急赶火车，只想要默认的套餐  order({});  //第三个用户: 只想换主食: 奥尔良烤腿堡，小吃和饮料保持  order({ zhushi:"奥尔良烤腿堡"});  //第四个用户: 想换主食和小吃: 巨无霸和鸡块, 饮料保持默认  order({  zhushi:"巨无霸",  xiaochi:"鸡块"  });    //第五个用户: 只想换小吃为辣翅，主食和饮料保持默认！  order({  xiaochi:"辣翅"  });    //第六个用户: 只想换饮料为奶茶，主食和小吃保持默认！  order({  yinliao:"奶茶"  });    //第七个用户: 想换小吃和饮料为土豆泥和红茶，主食保持不变！  order({  xiaochi:"土豆泥",  yinliao:"红茶"  })  </script>  运行结果:  您点的套餐是:  主食:香辣鸡腿堡,  小吃:薯条,  饮料:可乐  您点的套餐是:  主食:香辣鸡腿堡,  小吃:薯条,  饮料:可乐  您点的套餐是:  主食:奥尔良烤腿堡,  小吃:薯条,  饮料:可乐    您点的套餐是:  主食:巨无霸,  小吃:鸡块,  饮料:可乐    您点的套餐是:  主食:香辣鸡腿堡,  小吃:辣翅,  饮料:可乐    您点的套餐是:  主食:香辣鸡腿堡,  小吃:薯条,  饮料:奶茶    您点的套餐是:  主食:香辣鸡腿堡,  小吃:土豆泥,  饮料:红茶 |

三. class

对应小程序视频: 小程序->在线->JSCORE->day08 2. class

1. 什么是: 集中保存一个类型的构造函数和原型对象方法的程序结构

2. 为什么: 旧js中，同属于一个类型的构造函数和原型对象方法虽然是一家人，但是却分着写！！！不符合封装的要求！

3. 何时: 今后只要创建一种类型，包含构造函数和原型对象方法时，都要用class包裹构造函数和原型对象方法

4. 如何定义class: 3句话

(1). 用class{}包裹构造函数和原型对象方法

(2). 类型名不应该是属于构造函数，应该属于整个class，所以要将构造函数名/类型名提升到class后，作为整个class的名称。且所有构造函数从此统一更名为"constructor"

(3). 直接定义在class下的函数，默认就会保存到原型对象中成为共有方法，且不用再加"类型名.prototype"前缀，也不用在写"=function"

class 类型名{

constructor(形参列表){

this.属性名=形参;

... = ...

}

共有方法(){

... ...

}

}

5. 如何使用class: class就是新瓶装旧酒，底层还是以前的构造函数和原型对象，所以class的用法和构造函数的用法完全一样！

var 对象名=new 类型名(属性值列表)

6. 示例: 定义一个学生类型的class，并创建一个学生对象lilei

9\_class.html

|  |
| --- |
| <script>  //定义一种学生类型，规定所有学生的统一属性结构，要求所有学生都会做自我介绍  class Student{  //回顾: 一种类型都包括: 构造函数和原型对象  //先定义构造函数:  constructor(sname, sage){  this.sname=sname;  this.sage=sage;  }  //再向原型对象中添加共有方法  intr(){  console.log(`I'm ${this.sname},I'm ${this.sage}`)  }  }  //class就是新瓶装旧酒，底层还是以前的构造函数和原型对象，所以class的用法和构造函数的用法完全一样！  //想创建一个学生对象lilei  var lilei=new Student("Li Lei",11);  lilei.intr();  console.log(lilei);  </script>  运行结果:  I'm Li Lei,I'm 11  Student {sname: "Li Lei", sage: 11}  sage: 11  sname: "Li Lei"  \_ \_proto\_ \_:  constructor: class Student  intr: ƒ intr()  \_ \_proto\_ \_: Object |

7. 如何设置共有属性值:

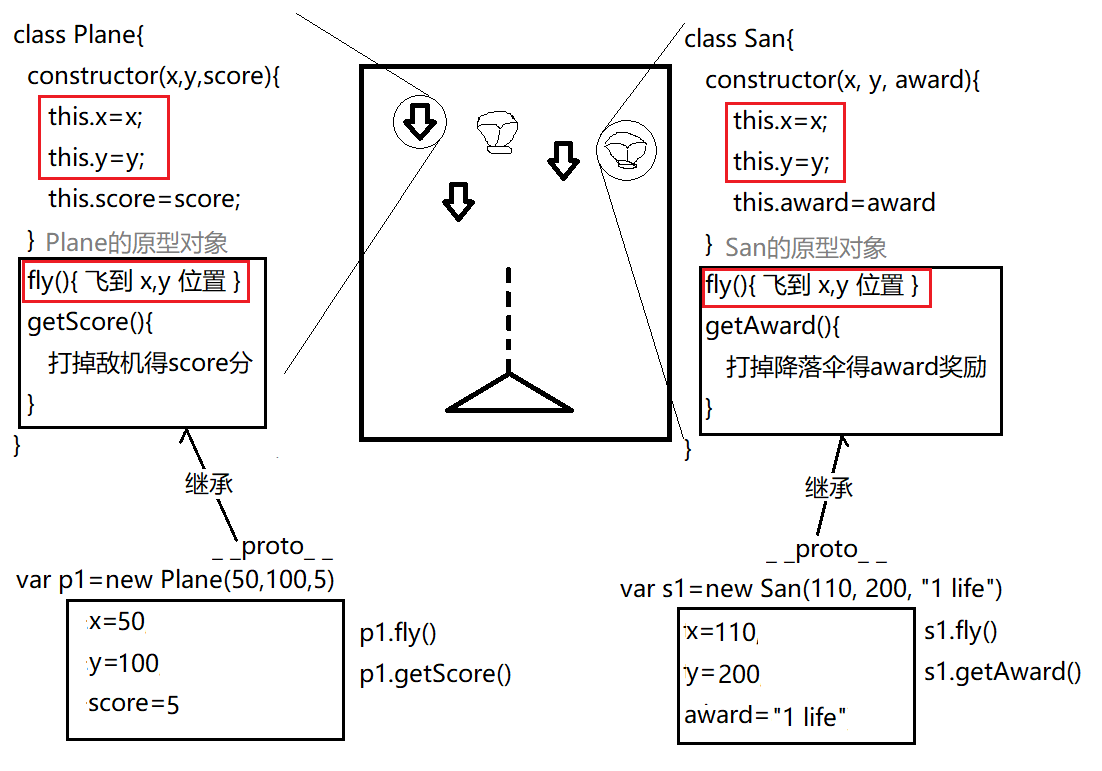
(1). 问题: 直接放在class中的属性值会自动复制到每个子对象中成为自有属性，所以class中就不应该出现属性值的定义，只包含构造函数和方法名

(2). 如何为所有子对象设置共有属性:

类型名.prototype.共有属性=属性值

8. 两种类型间的继承: extends

(1). 问题: 多个class中包含部分相同的属性结构和方法定义



(2). 解决: 再多定义一个父级class，集中保存两个子级class中相同部分的属性和方法。最后再让两个子级class继承父级class。

(3). 如何:

a. 先再多定义一个父级class，替两个子级class集中保存相同的属性和方法

b. 子级class中不需要包含重复的属性和方法

c. 子级class继承父级class:

class 子级 extends 父级class{

constructor(xxx){

super(xxx) //借用父级class中构造函数的意思

this.属性名=形参

}

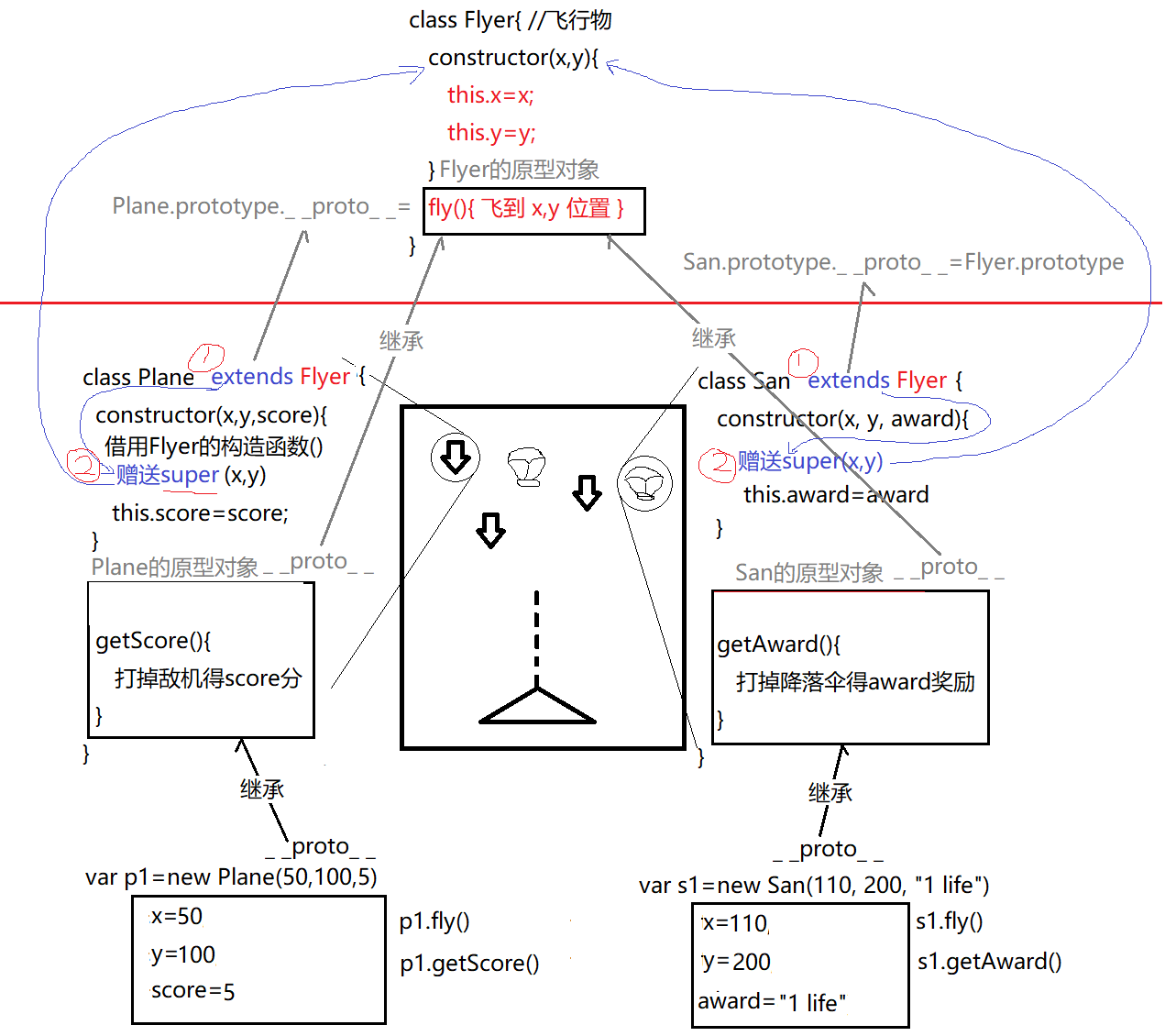
}

其中: extends简化了"子类型.prototype.\_\_proto\_\_=父类型.prototype"

并且还赠送了一个super()函数，专门代表父类型中没名的构造函数

调用super()等效于调用父级class的构造函数中的代码。

(4). 结果: 创建子级class的子对象，子对象既可使用父级class的属性和方法，又可使用爷爷级class的属性和方法。



(5). 示例: 定义飞机大战中飞机和降落伞类型，并使用继承避免重复代码

|  |
| --- |
| <script>      //先定义父类型Flyer      class Flyer{        constructor(x,y){          this.x=x;          this.y=y;        }        fly(){          console.log(`飞到x:${this.x},y:${this.y}位置`)        }      }      //再定义子类型继承父类型      class Plane extends Flyer{        constructor(x,y,score){          super(x,y);//必须写在子类型属性之前          this.score=score;        }        getScore(){          console.log(`击落敌机得${this.score}分`)        }      }      var p1=new Plane(50,100,5);      console.log(p1);      p1.fly();      p1.getScore();      //模仿Plane，自己定义一个San类型继承Flyer    </script>  运行结果:  Plane {x: 50, y: 100, score: 5}  score: 5  x: 50  y: 100  \_ \_proto\_ \_: Flyer  constructor: class Plane  getScore: ƒ getScore()  \_ \_proto\_ \_:  constructor: class Flyer  fly: ƒ fly()  \_ \_proto\_ \_: Object  飞到x:50,y:100位置  击落敌机得5分 |

总结:

\*\*\*\*ES6\*\*\*\*

1. 今后，拼接字符串都用模板字符串代替+号拼接

2. 今后，凡是声明变量都用let代替var。(不会被声明提前，不会超出代码块{})

3. 今后，几乎所有function都可用=>简写

特例: 只有不希望函数内this与函数外this保持一致时，才不能用箭头函数简写

4. 今后，凡是从头到尾遍历索引数组全部元素时，且只关心元素内容不关心元素位置的遍历都可用for of简写普通for循环

5. 今后，只要希望调用者没有传入实参值时，形参也有默认值可用就用：形参=参数默认值

6. 今后，只要希望定义函数时，可收集多个不确定个数的实参值时，都用...数组名，代替arguments。

7. 今后，只要希望调用函数时，先打散数组为多个值，再分别传参给函数时，都用...数组名

8. 今后，只要希望提取出数组或对象中个别成员值放在变量中单独简化使用时，就用解构。

var [变量1, 变量2, ...]=数组

var {属性名1, 属性名2, ...}=对象

9. 今后，只要多个实参值不确定有没有，但是又要求实参值必须对应的传给指定的形参变量时，就用参数解构:

(1). 定义函数时:

function 函数名({

形参名1=默认值1,

形参名2=默认值2,

... = ...

}){

... 函数体 ...

}

(2). 调用函数时:

函数名({

形参名1: 实参值1,

形参名2: 实参值2,

... : ...

})

10. 今后，只要定义一种类型包含构造函数和原型对象方法，都要用class包裹

class 类型名{

constructor(形参列表){

this.属性名=形参;

... = ...

}

共有方法(){

... ...

}

}

11. 今后，只要两种class中包含部分相同的属性和方法定义，都可用两种类型间的继承;

class 父类型{

constructor(形参列表){

this.相同部分的属性=形参

... = ...

}

相同部分的方法(){ ... }

}

class 子类型 extends 父类型{

constructor(形参列表){

supre(形参列表);

this.子类型独有属性=形参

... = ...

}

子类型独有方法(){ ... }

}

总结: this 判断this一定不要看定义在哪儿！只看调用时.前是谁或有没有new

4种:

1. obj.fun() this->.前的obj

2. new Fun() this->new正在创建的新对象

3. 构造函数.prototype.共有方法=function(){ ... } this->将来调用这个共有方法的.前的某个子对象

4. fun() 或 (function(){ ... })() 或 回调函数 this->window

今日对应小程序视频:

小程序->在线->JSCORE->day07 5. 参数增强 ...

6. 解构(上) ...

小程序->在线->JSCORE->day08 1. 解构(下) ...

2. class

作业:

1. 复习今日问题清单(小程序->首页->JSCORE->day07、day08)，预习明日问题清单(小程序->首页->JSCORE->day08)

2. 看小程序视频学习ES6中关于对象和属性的简写:

小程序->在线->JSCORE->day07 作业: ES6中关于对象中属性和方法的简写

3. 看小程序视频学习如何使用回调函数让多个异步函数顺序执行

小程序->在线->JSCORE->day07 作业: 如何使用回调函数让多个异步函数顺序执行

4. 看小程序视频预习promise

小程序->在线->JSCORE->day08 3. 回调地狱 callback hell 和promise...

5. (学有余力！！！)看小程序视频学习let+for相关笔试题:

小程序->在线->JSCORE->day07 作业: 高频笔试题 定时器setTimeout let 5个5

作业: 高频笔试题 多个按钮 点那个按钮 ...

6. (学有余力！！！)看小程序视频学习高频笔试题数组去重复

小程序->在线->JSCORE->day07 作业: 高频笔试题 数组去重复 Set类型