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->day05、day06

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->day05、day06

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->day05、day06

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，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16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: \*\*\*\*ES5\*\*\*\*

1. 保护对象

2. Object.create()

3. 替换this

4. 数组函数

一. 保护对象:

1. 问题: js中的对象自己毫无自保能力。别人可以随意修改对象的属性值和对象结构。对象对此毫无抵抗力。但是企业中一个对象的数据和结构往往不能随意修改为无意义的值。

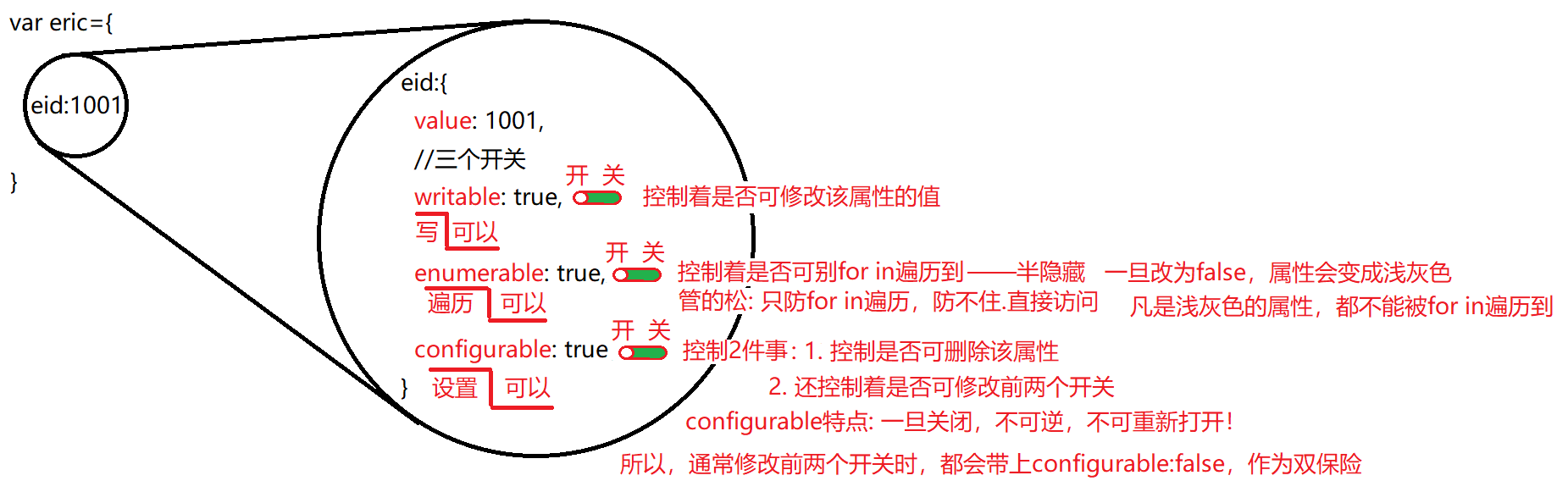
2. 解决: 在ES5标准中，其实已经给每个对象内置了自我保护的办法！

3. 如何保护对象：保护对象中的单个属性 + 保护整个对象的结构

4. 保护属性:

对应小程序视频: 小程序->在线->JSCORE->day05 4. 保护对象属性 开关...

(1). ES5标准中规定，对象中的一个属性，已经不再是一个普通的变量了。而是一个缩微的小对象



(2). 强调: 虽然对象的每个属性在底层也是一个缩微的小对象。但是小对象中的开关~~不能用.直接访问~~到！必须用专门的函数！

(3). 查看一个对象的属性的小对象中的开关值: （用的少，了解即可）

var obj=Object.getOwnPropertyDescriptor(对象, "属性名");

获得自己的属性的描述信息

(4). 修改一个对象中一个属性中开关的状态: ~~也不能用.直接修改~~

a. 如果只修改一个属性的开关:

(重新)定义属性

Object.defineProperty(对象名, "属性名", {

开关名: true或false,

开关名: true 或false

... ...

})

b. 示例: 使用defineProperty()修改属性的开关，保护属性

|  |
| --- |
| <script>  "use strict";  //一个员工对象eric  var eric={  eid:1001, //员工编号  /\*eid:{  value:1001,  writable:true, //控制着是否可修改属性值  enumerable:true, //控制着是否可被for in遍历到，但是防不住.直接访问  configurable:true //控制着: 1. 是否可删除该属性  // 2. 是否可修改前两个开关  }\*/  ename:"埃里克", //员工姓名  salary:12000 //薪资  }  //错误: 尝试用.访问每个属性小对象内的开关值  // console.log(  // eric.eid.value,  // eric.eid.writable,  // eric.eid.enumerable,  // eric.eid.configurable  // ); //undefined undefined undefined undefined  //正确：用专门的函数查看一个对象中一个属性的开关值  var obj=Object.getOwnPropertyDescriptor(eric,"eid");  console.log(obj);  //企业中有三个规定:  //1. 员工编号禁止随意修改——只读 eid的writable改为false  //2. 员工不能没有姓名属性——不能删除 ename的configurable改为false  //3. 员工的薪资不能随意被for in遍历 salary的enumerable改为false  //让eric的eid属性只读  //重新定义eric的eid属性的writable为false  Object.defineProperty(eric,"eid",{  writable:false,  configurable:false //双保险: 禁止修改前两个开关 —— 一旦关闭不可逆！  })  //让eric的ename禁止删除  //重新定义eric的ename属性的configurable为false  Object.defineProperty(eric,"ename",{  configurable:false  })  //不想让eric的salary随意被for in遍历  //重新定义eric的salary属性的enumerable:false  Object.defineProperty(eric,"salary",{  enumerable:false,  configurable:false //双保险  })  //屌丝: 尝试将已经关闭的eid的writable重新打开，再修改  // Object.defineProperty(eric,"eid",{  // writable:true,  // configurable:true  // })//Cannot redefine property: eid  // // 不能 重新定义 属性 eid  //试图修改eid  // eric.eid=-2; //报错:  // //Cannot assign to read only property 'eid'  // //不能 赋值 给 只读 属性 eid  //试图删除ename  //删除 对象的属性——危险  //delete eric.ename; //报错: Cannot delete property 'ename'  //试图遍历eric所有属性，包括工资！  for(var key in eric){  console.log(`属性名:${key}, 属性值: ${eric[key]}`)  }  console.log(eric);  //尝试用.直接访问eric的salary属性  console.log(`薪资:${eric.salary}`) //依然可以正常输出薪资  </script>  运行效果:  {value: 1001, writable: true, enumerable: true, configurable: true}  属性名:eid, 属性值: 1001  属性名:ename, 属性值: 埃里克  {eid: 1001, ename: "埃里克", salary: 12000}  薪资:12000 |

c. 问题: 因为一个对象中肯定有多个属性都需要保护，但是defineProperty一次只能修改一个属性内的开关。如果想修改多个属性中的开关，就要反复写很多遍——麻烦！

d. 解决: 用defineProperties()一次修改多个属性内的开关

Object.defineProperties(对象名,{

属性名:{

开关: true或false,

... : ... ,

},

属性名:{

开关: true或false,

... : ... ,

},

... ...

})

e. 示例: 使用defineProperties()简化defineProperty()

|  |
| --- |
| <script>  "use strict";  //一个员工对象eric  var eric={  eid:1001, //员工编号  /\*eid:{  value:1001,  writable:true, //控制着是否可修改属性值  enumerable:true, //控制着是否可被for in遍历到，但是防不住.直接访问  configurable:true //控制着: 1. 是否可删除该属性  // 2. 是否可修改前两个开关  }\*/  ename:"埃里克", //员工姓名  salary:12000 //薪资  }  //错误: 尝试用.访问每个属性小对象内的开关值  // console.log(  // eric.eid.value,  // eric.eid.writable,  // eric.eid.enumerable,  // eric.eid.configurable  // ); //undefined undefined undefined undefined  //正确：用专门的函数查看一个对象中一个属性的开关值  var obj=Object.getOwnPropertyDescriptor(eric,"eid");  console.log(obj);  //企业中有三个规定:  //1. 员工编号禁止随意修改——只读 eid的writable改为false  //2. 员工不能没有姓名属性——不能删除 ename的configurable改为false  //3. 员工的薪资不能随意被for in遍历 salary的enumerable改为false  //让eric的eid属性只读  //重新定义eric的eid属性的writable为false  //让eric的ename禁止删除  //重新定义eric的ename属性的configurable为false  //不想让eric的salary随意被for in遍历  //重新定义eric的salary属性的enumerable:false  Object.defineProperties(eric,{  eid:{  writable:false,  configurable:false  },  ename:{  configurable:false  },  salary:{  enumerable:false,  configurable:false  }  })  //屌丝: 尝试将已经关闭的eid的writable重新打开，再修改  // Object.defineProperty(eric,"eid",{  // writable:true,  // configurable:true  // })//Cannot redefine property: eid  // // 不能 重新定义 属性 eid  //试图修改eid  // eric.eid=-2; //报错:  // //Cannot assign to read only property 'eid'  // //不能 赋值 给 只读 属性 eid  //试图删除ename  //删除 对象的属性——危险  //delete eric.ename; //报错: Cannot delete property 'ename'  //试图遍历eric所有属性，包括工资！  for(var key in eric){  console.log(`属性名:${key}, 属性值: ${eric[key]}`)  }  console.log(eric);  //尝试用.直接访问eric的salary属性  console.log(`薪资:${eric.salary}`) //依然可以正常输出薪资  </script>  运行结果:  {value: 1001, writable: true, enumerable: true, configurable: true}  属性名:eid, 属性值: 1001  属性名:ename, 属性值: 埃里克  {eid: 1001, ename: "埃里克", salary: 12000}  薪资:12000 |

(5). 访问器属性:

对应小程序视频: 小程序->在线->JSCORE->day05 5. 访问器属性 get set ...

a. 问题: 仅使用开关无法用自定义规则灵活保护属性

b. 解决: 给对象中的一个属性请保镖——访问器属性

c. 什么是访问器属性: 属性自己不实际保存属性值，仅提供对另一个属性的保护功能，这样的特殊属性，称为访问器属性，俗称保镖

d. 何时: 今后只要想用自定义规则灵活保护属性时，都要用访问器属性

e. 如何: 3步: ~~访问器属性不能在对象{}内直接写！~~

只能通过Object.defineProperty()或Object.defineProperties()添加

Object.defineProperties(对象, {

1). 先定义一个隐姓埋名且半隐藏的属性准备实际保存属性值

\_属性名:{

value: 属性值,

writable:true, //这种属性值往往可以改，只不过需要验证通过才能改

enumerable:false, //不想让受保护的属性随意被for in遍历到

configurable:false, //以上设置禁止随意修改

},

2). 为要保护的属性: \_属性名 请保镖：

//保镖要冒名顶替\_属性名

对外公开属性名:{ //保镖一请就是一对儿

//专门负责从受保护的\_属性中获取属性值给外部

get:function(){

return this.\_属性名; //从受保护的\_属性中读取属性值

},

//专门负责将要修改的新属性值验证后保存到受保护的\_属性中

set:function(value){

//形参value: 自动接住要修改给属性的新值

if(value符合条件){

this.\_eage=value; //将value新值真正保存到受保护的\_属性中

}else{

throw Error("报错")

}

},

enumerable:true,//保镖要代替\_属性抛头露面，所以应该for in可看到

configurable:false, //以上设置禁止随意修改

//因为保镖不实际存值，所以没有value属性

//因为writable不好用，我们才用的访问器属性，所以一旦用get()和set()就不再需要使用writable开关了。等于用灵活的get()和set()代替了不灵活的writable。

}

})

3). 对象外或其他人使用对象内的访问器属性和使用普通属性完全一样！因为外界根本就不知道他们访问的是一个冒名顶替的访问器属性。蒙在鼓里。

i. 想获取访问器属性值时:

console.log(对象名.访问器属性名)

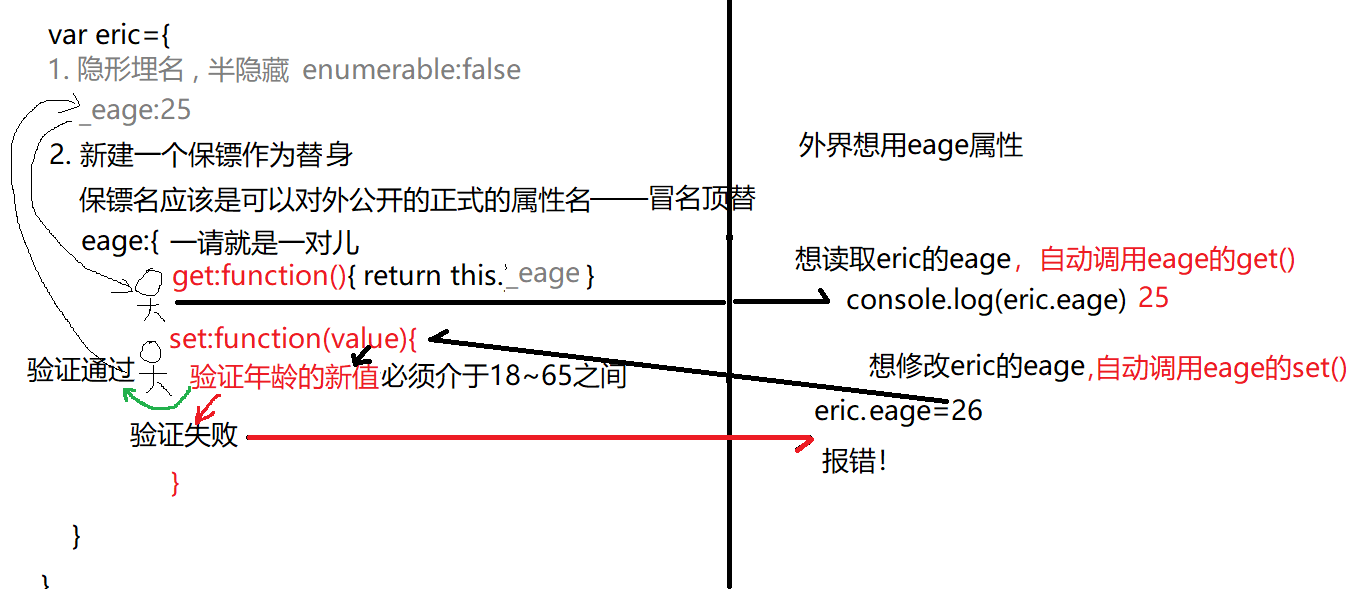
自动调用访问器属性中的get()，从\_属性中获取属性值返回给外部

ii. 想修改访问器属性值时:

对象名.访问器属性名=新属性值

自动调用访问器属性中的set()，并自动将=右侧的新属性值传给set()的形参value。

set()内验证value是否符合要求。如果value符合要求，才将value实际保存到受保护的\_属性中。如果value不符合要求，就报错，并且不会把错误的value保存到\_属性中。

s

f. 示例: 使用访问器属性保护年龄属性: 年龄必须介于18~65之间

|  |
| --- |
| <script>  var eric={  eid:1001,  ename:"埃里克",  eage:25  };  //年龄eage可以修改，但是年龄必须介于18~65之间！  //1. 先将eage属性值保存到一个半隐藏的\_eage属性中  Object.defineProperties(eric,{  \_eage:{//受保护的实际保存数据的属性  value:eric.eage, //将旧eric的eage属性值转义到半隐藏的新的\_eage属性中  writable:true,//年龄可以改  enumerable:false, //不想让受保护的属性随意被for in遍历到  configurable:false, //以上设置禁止随意修改  },  //2. 为\_eage请保镖:  //保镖要冒名顶替eage  eage:{ //保镖一请就是一对儿  //专门负责从受保护的\_eage属性中获取属性值给外部  get:function(){  console.log(`当试图读取eage的属性值时，自动调用一次eage的get()`)  //this->当前eric对象  return this.\_eage;  },  //专门负责将要修改的新属性值验证后保存到受保护的属性中  set:function(value){  console.log(`当试图修改eage的属性值时，自动调用一次eage的set(value=${value})`);  if(value>=18&&value<=65){  this.\_eage=value;  }else{  throw Error("您的年龄超范围，无法保存！")  }  },  enumerable:true,//保镖要代替\_eage属性抛头露面  configurable:false, //以上设置禁止随意修改  //因为保镖不实际存值，所以没有value属性  //因为writable不好用，我们才用的访问器属性，所以一旦用get()和set()就不再需要使用writable开关了  }  });  //外界: 使用访问器属性和使用普通属性完全一样！因为外界根本不知道自己访问的是访问器属性。  //想获取eric的年龄  console.log(eric.eage);//  //先修改eric的年龄  eric.eage=26;//  console.log(eric);  //试图修改错误的年龄  eric.eage=-2;  </script>  运行结果:  当试图读取eage的属性值时，自动调用一次eage的get()  25  当试图修改eage的属性值时，自动调用一次eage的set(value=26)  {eid: 1001, ename: "埃里克", \_eage: 26}  当试图修改eage的属性值时，自动调用一次eage的set(value=-2)  Uncaught Error: 您的年龄超范围，无法保存！  at Object.set [as eage] (3\_get\_set.html:39)  at 3\_get\_set.html:56 |

5. 保护对象的结构: 3级别

对应小程序视频: 小程序->在线->JSCORE->day06 1. 保护对象结构

(1). 防扩展: 禁止给对象随意添加新属性

a. 旧js中: 对象即使创建完之后，依然可以通过强行赋值方式为对象随意添加新属性

b. 如何防扩展: Object.preventExtensions(对象)

c. 意为: 阻止(添加)扩展(给)对象

d. 示例: 阻止为eric对象添加新属性

|  |
| --- |
| <script>  "use strict";  var eric={  eid:1001, //只读  ename:"埃里克"  };  //设置eid只读  Object.defineProperty(eric,"eid",{  writable:false,  configurable:false  });  //禁止给eric对象扩展新属性  Object.preventExtensions(eric);  //尝试修改eid  //eric.eid=-2; //报错:Cannot assign to read only property 'eid'  // 不能 赋值 给 只读 属性 eid  //但是，因为eid管的太严，屌丝觉得不好用，于是自己另起炉灶，添加了一个甚至多个eid  //eric.Eid=-2; //报错:  //Cannot add property Eid, object is not extensible  //不能 添加 属性 Eid（因为）对象是不 可扩展  console.log(eric);  </script>  运行结果:  Uncaught TypeError: Cannot add property Eid, object is not extensible  at 4\_preventExtensions.html:28 |

(2). 密封: 既禁止添加新属性，又禁止删除现有属性

a. 问题: 对象中几乎所有属性都应该是禁止删除的，那就意味着每个属性都要添加configurable:false开关。——麻烦！

b. 解决: 只要密封对象即可

c. 如何: Object.seal(对象)

密封

d. 意为: 既禁止给对象添加新属性，又禁止删除对象中现有属性

e. 原理: seal()内部自动做了两件事:

1). 自动调用了Object.preventExtensions()——防扩展

2). 自动将所有属性的configurable都改为了false

f. 好处: 从此所有属性的configurable:false，都可省略！

g. 何时: 今后几乎所有对象都应该密封起来！

h. 示例: 密封eric对象

|  |
| --- |
| <script>  "use strict";  var eric={  eid:1001, //只读  ename:"埃里克", //禁止删除  salary:12000 //禁止for in遍历  }  Object.defineProperties(eric,{  eid:{writable:false},  salary:{enumerable:false}  });  //密封eric对象  Object.seal(eric);  //尝试重新打开eid的writable开关  // Object.defineProperty(eric,"eid",{  // writable:true,  // configurable:true  // });//报错: Cannot redefine property: eid  //尝试修改eid的值  // eric.eid=-2;//报错: Cannot assign to read only property 'eid'  //尝试删除ename属性  // delete eric.ename;//报错: Cannot delete property 'ename'  //尝试遍历eric中所有属性  for(var key in eric){  console.log(`属性名:${key},属性值:${eric[key]}`)  }  </script>  运行结果:  Uncaught TypeError: Cannot redefine property: eid  at Function.defineProperty (<anonymous>)  at 5\_seal.html:26 |

(3). 冻结: 既禁止添加新属性，又禁止删除现有属性，甚至连属性值都不能修改了

a. 如何: Object.freeze(对象)

b. 原理: 悄悄做了三件事:

1). 自动调用了Object.preventExtensions()——防扩展

2). 自动将所有属性的configurable都改为false——密封

3). 自动将所有属性的writable都改为false

c. 何时:

1). 大部分对象不用冻结  
 2). 少量多个模块共用的对象，不想因为某一个模块篡改了对象中的内容，而导致牵一发而动全身时，都要将这个共用的对象冻结！（连属性值都不能个修改！）

d. 示例: 冻结一个对象，禁止修改其属性值

|  |
| --- |
| <script>  "use strict"  var pool={  host:"127.0.0.1",//writable:false  port:3306,//writable:false  db:"xz"//writable:false  }  Object.freeze(pool);  pool.host="192.168.0.100";//报错: Cannot assign to read only property 'host'  </script>  运行结果:  Uncaught TypeError: Cannot assign to read only property 'host' of object '#<Object>'  at 6\_freeze.html:19 |

二.Object.create()

1. 什么是Object.create(): 专门用于根据一个现有父对象，临时创建一个新的子对象继承当前父对象。

2. 问题: 通常创建子对象都是new 构造函数()，但是如果没有构造函数，只有父对象时，也想创建子对象，继承父对象，就无法用new了！

3. 解决: 用Object.create()临时创建子对象

4. 何时: 即使没有构造函数，只有一个父对象，也想创建子对象继承父对象时，就可用Object.create()

5. 如何:

var 子对象名=Object.create(父对象,{

属性名:{

value:属性值,

开关: true 或 false

},

属性名:{

value:属性值,

开关: true 或 false

},

... ...

})

6. Object.create()一共做了3件事:

(1). 创建一个新的空对象

(2). 让新对象继承()中第一个参数对象father

(3). 将()中第二个对象中的属性添加到新对象中成为新对象的自有属性

7. 示例: 没有构造函数，只有父对象，也想创建子对象继承父对象

|  |
| --- |
| <script>  //只有父对象，没有构造函数  var father={  bal:100000000000,  car:"infiniti"  }  //也想临时创建子对象继承father  var lilei=Object.create(father,{  //不能直接写属性值，必须按照defineProperties的格式写  sname:{  value:"Li Lei",  writable:true,  enumerable:true  },  sage:{  value:11,  writable:true,  enumerable:true  }  });//3件事  Object.seal(lilei);//密封  //1. 创建一个新的空对象  //2. 让新的空对象继承father对象  //3. 为新对象添加自有属性  console.log(lilei);  console.log(lilei.sname, lilei.sage, lilei.bal, lilei.car);  </script>  运行结果:  {sname: "Li Lei", sage: 11}  sage: 11  sname: "Li Lei"  \_ \_proto\_ \_:  bal: 100000000000  car: "infiniti"  \_\_proto\_\_: Object  Li Lei 11 100000000000 infiniti |

三. 替换this:

对应小程序视频: 小程序->在线->JSCORE->day06 2. 替换this call,apply,bind

1. 其实如果一个函数中的this不是想要的！是可以随意更换的！

2. 2种情况:

(1). 在一次调用函数时，临时替换一次this为指定的对象

a. 如何: 要调用的函数.call(替换this的对象, 实参值1, 实参值2,...)

b. 结果:

1). 在本次调用函数时，函数中的this会被临时替换为()中指定的任意对象

2). 将从第二个实参开始往后的所有实参值自动传给要调用的函数的形参变量

c. 示例: 使用call替换薪资计算函数中的this，计算不同员工的薪资

|  |
| --- |
| <script>  //有一个公共的计算薪资的函数  //三个形参 底薪 奖金1 奖金2  function jisuan(base, bonus1, bonus2){  console.log(`${this.ename} 的总工资是: ${base+bonus1+bonus2}`)  }  //有两个员工:  var lilei={ ename:"Li Lei" };  var hmm={ ename:"Han Meimei" };    //李磊这个月底薪10000，奖金1:1000, 奖金2:2000，李磊想用jisuan()计算自己的总工资  //错误做法1:  //jisuan() //this->window window.ename->undefined  //错误做法2: lilei能调用全局函数jisuan()  //console.log(lilei); //查看李磊的原型链都没有jisuan()  //因为jisuan()函数保存在全局window中，不属于任何一个对象的原型链，只属于作用域链。  //lilei.jisuan() //报错: lilei.jisuan is not a function  //正确做法:  jisuan.call(lilei,10000,1000,2000)  // | ↓ ↓ ↓  //jisuan( ↓ base,bonus1,bonus2)  // this.ename ....  //call()会临时将jisuan()中的this->lilei  //this.ename -> lilei.ename -> "Li Lei"  //又能将10000,1000,2000依次传给jisuan()函数的base,bonus1,bonus2  //如果hmm也想用jisuan()计算自己的薪资:底薪4000,奖金1:5000,奖金2:6000  jisuan.call(hmm,4000, 5000, 6000)  // | ↓ ↓ ↓  // jisuan( ↓ base,bonus1,bonus2)  // this.ename...  </script>  运行结果：  Li Lei 的总工资是: 13000  Han Meimei 的总工资是: 15000 |

d. 特殊情况: 如果要调用的函数需要传入多个实参值，但是给定的实参值却是放在一个数组中给的。出现了不一致！

e. 解决: 用apply()代替call，将实参值数组直接传入apply()

要调用的函数.apply(替换this的对象, 保存实参值的数组)

f. 原理: apply比call多一项功能，可自动打散数组为多个实参值，再依次分别传给函数的形参变量。

g. 示例: 使用apply()代替call()，先打散数组，再传参

|  |
| --- |
| <script>  //有一个公共的计算薪资的函数  //三个形参 底薪 奖金1 奖金2  function jisuan(base, bonus1, bonus2){  console.log(`${this.ename} 的总工资是: ${base+bonus1+bonus2}`)  }  //一个员工lilei  var lilei={ ename:"Li Lei" };  //李磊想用jisuan()计算自己的薪资  //但是李磊的薪资是放在一个数组中的  var arr=[10000,1000,2000];  //jisuan.call(lilei,arr[0],arr[1],arr[2])  jisuan.apply(lilei,arr)  // | 打散数组为多个值  // | 10000 1000 2000  // | ↓ ↓ ↓  // jisuan( ↓ base,bonus1,bonus2)  // this.ename  </script>  运行结果：  Li Lei 的总工资是: 13000 |

(2). 基于原函数创建一个新函数副本，并永久替换其中的this

a. 问题: 如果一个函数每次调用时都要用call()替换this——麻烦

b. 解决: 创建一个新函数副本，并永久绑定新函数中的this为指定对象

c. 如何: var 新函数=原函数.bind(替换this的对象, 要固定的实参值)

d. 原理: bind共做了3件事:

1). 基于原函数创建一个一模一样的函数副本

2). 永久替换新函数副本中的this为指定对象

3). 永久绑定个别固定的实参值

e. 示例: 基于原函数创建一个新函数副本，并永久绑定this

|  |
| --- |
| <script>  //有一个公共的计算薪资的函数  //三个形参 底薪 奖金1 奖金2  function jisuan(base, bonus1, bonus2){  console.log(`${this.ename} 的总工资是: ${base+bonus1+bonus2}`)  }  //一个员工lilei  var lilei={ ename:"Li Lei" };  //每个月开工资时都要计算自己的薪资  //李磊买一个自己专属的新计算器呢！  var jisuan2=jisuan.bind(lilei,10000);  //2件事:  //创建了一个和jisuan()一模一样的新函数副本  //将新函数jisuan2中的this永久替换为了lilei  //将将函数中第一个形参base永久固定为10000  // 底薪几乎不变  jisuan2(1000,2000);  jisuan2(500,1000);  jisuan2(2000,2000);  </script>  运行结果:  Li Lei 的总工资是: 13000  Li Lei 的总工资是: 11500  Li Lei 的总工资是: 14000 |

四. 数组新增函数:

1. 判断: 2个

对应小程序视频: 小程序->在线->JSCORE->day06 3. every原理 判断数组...

(1). 判断数组中是否所有元素都符合要求

// 每个/所有

a. var bool=arr.every(function(elem, i, arr){

return 判断结果

})

b. 原理:

1). every内自带for循环遍历数组中每个元素——信任

2). 每遍历到一个元素就自动调用一次回调函数——信任

3). 每次调用回调函数时都自动传入三个实参值:

i. 将当前正在遍历的一个元素值交给第一个形参elem

ii. 将当前正在遍历到的下标位置交给第二个形参i

iii. 将当前every前的.前的数组对象交给第三个形参arr

4). 在回调函数内部，利用传入的三个实参值判断当前元素是否符合要求，并返回当前元素的判断结果

5). every要求:

i. 所有元素的判断结果都返回true时，整个every才返回true，说明整个数组都是符合要求的。

ii. 反之，如果遍历过程中，碰到一个元素的判断结果返回false，则没必要再向后执行。而是整个every直接返回false，表示当前数组中不是所有元素都符合要求！

c. 示例: 判断数组是否全由奇数组成:

|  |
| --- |
| <script>  var arr1=[1,2,3,4,5]; //false  var arr2=[1,3,5,3,1]; //true  //想判断哪个数组全由奇数组成:  var result1=arr1.every(function(elem,i,arr){  console.log(`arr1.every()自动执行一次回调函数: elem=${elem},i=${i},arr=${arr},本次判断结果: ${elem%2==1}`);  //每个元素都要是奇数才行  return elem%2==1 //只判断当前一个元素是不是奇数  });  var result2=arr2.every(function(elem,i,arr){  console.log(`arr2.every()自动执行一次回调函数: elem=${elem},i=${i},arr=${arr},本次判断结果: ${elem%2==1}`);  return elem%2==1  });  console.log(result1,result2);  </script>  运行结果: |

(2). 判断数组中是否包含符合要求的元素

// 有一些

a. var bool=arr.some(function(elem,i,arr){

return 判断条件

})

b. 原理:

1). some内自带for循环遍历数组中每个元素——信任

2). 每遍历到一个元素就自动调用一次回调函数——信任

3). 每次调用回调函数时都自动传入三个实参值:

i. 将当前正在遍历的一个元素值交给第一个形参elem

ii. 将当前正在遍历到的下标位置交给第二个形参i

iii. 将当前every前的.前的数组对象交给第三个形参arr

4). 在回调函数内部，利用传入的三个实参值判断当前元素是否符合要求，并返回当前元素的判断结果

5). some要求:

i. 只要碰到一个元素检查结果为true，就不必再继续，整个some就返回true，说明数组中包含符合要求的元素。但是包含几个，不知道！

ii. 反之，如果some遍历结束都没有碰到一个检查结果为true的元素，整个some才被迫返回false，说明整个数组中都不包含任何符合要求的元素。

c. 示例: 判断哪个数组中包含偶数:

|  |
| --- |
| <script>  var arr1=[1,2,3,4,5]; //true  var arr2=[1,3,5,3,1]; //false  //想判断哪个数组包含偶数  var result1=arr1.some(function(elem,i,arr){  console.log(`arr1.some()自动执行一次回调函数: elem=${elem},i=${i},arr=${arr},本次判断结果: ${elem%2==0}`);  //如果当前元素是偶数就返回true  return elem%2==0  });  var result2=arr2.some(function(elem,i,arr){  console.log(`arr2.some()自动执行一次回调函数: elem=${elem},i=${i},arr=${arr},本次判断结果: ${elem%2==0}`);  //如果当前元素是偶数就返回true  return elem%2==0  });  console.log(result1,result2);  </script>  运行结果: |

2. 遍历

3. 过滤

4. 汇总

总结: 面向对象: 封装，继承，多态

1. 封装: 创建对象 2种方法  
 (1). 如果只创建一个对象:

var 对象名={

属性名: 属性值,

... : ... ,

方法名: function(){

... this.属性名 ...

}

}

(2). 反复创建多个相同结构的对象: 构造函数 2步

a. 定义构造函数:

function 类型名(形参变量列表){

this.属性名=形参;

... = ... ;

//构造函数中，不要再包含方法定义

}

b. 用new 调用构造函数，创建新对象:

var 对象名=new 类型名(属性值列表)

|  |
| --- |
| new做了4件事:  1. 创建新对象  2. 继承: 让新子对象自动继承构造函数的原型对象  3. 调用构造函数，替换构造函数中的this为new新创建的子对象，通过强行赋值将构造函数中规定的属性添加到新的子对象中  4. 返回新对象的地址保存到等号左边的变量里 |

2. 继承: 父对象的成员，子对象无需重复创建就可直接使用

(1). 何时: 只要多个子对象都要共用同一个方法或属性值时

(2). 如何: 通过强行赋值，将共用的方法或属性值，添加到构造函数的原型对象中

构造函数.prototype.共有属性名=属性值

构造函数.prototype.共有方法名=function(){

... this.属性名 ...

}

3. 多态(重写):

(1). 何时: 如果子对象觉得从父对象继承来的方法不好用，就可用重写。

(2). 如何: 只要在子对象中添加和父对象中成员同名的自有成员，从此子对象总是优先使用自己自有的成员，不再使用父对象中同名的成员。

4. 自定义继承:

(1). 何时: 如果整个父对象都不是想要的，就可以整体更换为想要的新的父对象

(2). 如何: 2种:

a. 只修改一个对象的父对象: 2种:

1). 子对象.\_ \_proto\_ \_=新父对象

2). Object.setPrototypeOf(子对象, 新父对象)

b. 同时修改将来多个子对象的父对象:

1). 其实就是修改构造函数的原型对象

构造函数.prototype=新对象

2). 时机: 必须在创建子对象之前更换

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(){ ... })() this->window

ES5:

1. 严格模式: "use strict"

(1). 禁止给未声明的变量赋值

(2). 静默失败升级为错误

(3). fun() 或 (function(){ ... })() this->undefined

(4). 禁用了arguments.callee

2. 保护对象:

(1). 保护对象属性: 2种:

a. 用开关保护对象的属性:

1). 每个属性有3个开关:

i. writable: true或false 控制是否可修改当前属性值

ii. enumerable: true 或false 控制是否可用for in遍历到该属性

半隐藏(浅紫色)，只防for in 不防.

iii. configurable: true或false 控制2件事:

①是否可删除该属性；②是否可修改前两个特性

一旦改为false 不可逆

2). 修改开关: 2种:

i. 只修改一个属性中的开关:

Object.defineProperty(对象,"属性名",{

开关: true或false,

configurable:false //双保险

})

ii. 同时修改多个属性中的开关:

Object.defineProperties(对象,{

属性名:{

开关: true或false,

configurable:false //双保险

},

属性名:{

开关: true或false,

configurable:false //双保险

},

... ...

})

b. 用自定义规则保护对象的属性: 访问器属性 3步:

Object.defineProperties(对象,{

1). 定义一个隐形埋名且半隐藏的普通属性实际保存属性值

\_属性名:{

value:属性值,

writable: true, //通常受保护的属性都可以修改，只不过满足条件才能改

enumerable:false, //半隐藏，不想被外人轻易发现

configurable:false //以上设置禁止随意修改

},

2). 请保镖: 冒名顶替原属性，一请就是一对儿

对外公开的属性名:{

get:function(){ //负责从受保护的\_属性中获取属性值

return this.\_属性名

},

set:function(value){ //负责将新属性值经过验证后更新到受保护的\_属性中

if(value符合要求){

this.\_属性名=value

}else{

throw Error("错误提示")

}

},

enumerable:true, //保镖要代替被保护的\_属性名抛头露面

configurable:false, //以上配置禁止随意修改

}

})

3). 使用访问器属性: 和使用普通属性是完全一样的。其它程序不知道自己用的是访问器属性——蒙在鼓里。

i. 想获取访问器属性值: 对象.访问器属性

程序会自动调用访问器属性内部的get()，返回受保护的\_属性名

ii. 想修改访问器属性值: 对象.访问器属性=新值

程序会自动调用访问器属性内部的set()，将新值自动传给value形参，如果value形参验证通过，才将新属性值value，保存到受保护的\_属性名中

(2). 保护对象的结构: 3个级别:

a. 防扩展: Object.preventExtensions(对象)

b. 密封: Object.seal(对象)

c. 冻结: Object.freeze(对象)

3. Object.create()

(1). 何时: 即使没有构造函数，只有一个父对象，也想创建子对象继承父对象时都用Object.create()

(2). 如何:

var 子对象名=Object.create(父对象,{

自有属性名:{

value:属性值,

writable: true 或false,

enumerable:true 或 false,

},

自有属性名:{

value:属性值,

writable: true 或false,

enumerable:true 或 false,

},

... ...

})

Object.seal(子对象);

4. 替换this: 3种情况:

(1). 只在一次调用函数时，临时替换一次函数中的this: call 或 apply

a. 多数情况下首先call()

b. 只有在多个实参值被放在一个数组中给定时，需要先打散数组，再传参：apply()

(2). 创建一个新函数并永久绑定this为一个指定的对象: bind()

今日课上对应视频列表:

小程序->在线->JSCORE->day05 4. 保护对象属性 开关...

5. 访问器属性 get set ...

小程序->在线->JSCORE->day06 1. 保护对象结构

2. 替换this call,apply,bind

3. every原理 判断数组是否全由偶数组成

作业:

1. 复习今日问题清单(小程序->首页->JSCORE->day05、day06)，预习明日问题清单(小程序->首页->JSCORE->day07)

2. 看小程序视频复习和预习新增数组函数：

小程序->在线->JSCORE->day06 3. every原理 ...

4. forEach

5. forEach 按值传递

6. map原理...

7. filter

8. reduce原理

作业: 数组加也有indexOf()

3. （学有余力）看小程序视频学习使用reduce统计一个字符串这种每种字符出现的次数

小程序->在线->JSCORE->day06 作业: 使用reduce统计一个字符串中每种字符出现的次数