1.原型是function对象的一个属性，它定义了构造函数制造出了对象的共有祖先。通过该构造函数产生的对象，

可以继承该原型的属性和方法。原型也是对象。

2.原型即prototype，函数刚出生原型就定义好了。它类似一个空对象。Person.prototype={}

它是继承关系如，

Person.prototype.name ="hehe"

function Person() {}

var person = new Person();

var person1 = new Person();

person.name和person1.name都是"hehe" 共有祖先。

3.自己身上有原型上一模一样的属性时出现自己身上的属性。

4.函数会继承原型上的属性。

如，Person.prototype.name ="hehe"也可以这么写 Person.prototype = {name : "hehe"}(原型类似一个空对象)

function Person() {

age = 18;

}

var person = new Person();

person访问person.name出现18，访问person.name出现"hehe"。

5.利用原型，可以提取出共有属性。

Car.prototype.height = 1400;

function Car(color) {

this.color = color

}

var car = new Car("red"); height是需要固定不变的。

6.原型的增删改查。

增

Car.prototype.height = 1400;

function Car(color) {

this.color = color

}

var car = new Car(）；

car.prototype.height = 1000;

删改查 和对象的差不多不过要加上prototype如上

7.constructor（即构造器）属性在原型的类似空对象里面

这个属性可以找到祖先。

constructor可以手动更改，修改后继承修改的那个原型。

8. \_命名规则有这个表示尽量别修改里面的东西，但一定要修改也可以（人为的从\_\_proto\_\_来的）。

9.\_\_proto\_\_里面存的是原型如果修改器指向对象就会修改继承对象。

10.基础规则引用值堆套用//

Person.prototype.name = "sunny";

function Person() {

//var this = {\_\_proto\_\_ : Person.prototype}构造函数第一步发生的

}

var person = new Person();

Person.prototype = {

name = "cherry"

}

打印出的结果为sunny

相当与

Person.prototype = {name : "a"}; var obj = {name : "a"};

\_\_proto\_\_ = Person.prototype; var obj1 = obj；

Person.prototype = {name : "b"}; obj = {name : "b"};

11.Person.prototype.name = "sunny";

function Person() {

//var this = {\_\_proto\_\_ : Person.prototype}构造函数第一步发生的

}

Person.prototype = {

name = "cherry"

}

var person = new Person();

打印出cherry. 因为预编译环节。functtion Person() {}先提升上去了

12.原型链的连接点就是prototype。 \_\_proto\_\_ ：prototype

13.object.prototype是所有原型链的终端。

14.原型链不能增删改查 只自己修改自己原型。

15.var obj = {} 等于var obj1 = new Object(); 一般用对象字面量var obj = {} 开发规范（因为好操作）

16.var obj的constructor是Object()就是原型链的顶端。

17.Object.creat(对象或者null)（对象的话原型就指向这个对象）

Object.creat(null); 现在对象不最终继承Object.prototype除了这个全是最终继承Object.prototype

18.人为设的\_\_proto\_\_不会继承（写了Object.creat(null)截断了继承对象，然后人为设个\_\_proto\_\_指向是不会继承的）

19.因为原始值undefind和null不是对象，也没有原型链所以不可以经过包装类所以不可以添加属性和方法。

原始值undefined和null没有toString（） 写会报错

20. .优先级很高

123.toString(); 不可以这样

需要var num = 123 num.toString();

num.toString 包装 new Number(num).toString;

21.同样的名字实现不同的功能叫重写。

String.prototypr.toString()重写了

Boolean.prototype.toString（）重写了

原始值除了undefined和null不能用toString方法的其他的toString方法都重写了。

-----------------------------------------------------------------------------------------

22.构造函数发生的第一步 //var this = {\_\_proto\_\_ : Person.prototype}构造函数第一步发生的 ，

function Person() {

age = 19;

}

var person = new Person(); 访问person.age打印不出19(需要添加到原型上才能访问出来)

23.\_\_proto\_\_和constructor是对象独有的属性（在js中对象是由函数构成的所以有这两个属性)

prototype是函数独有的属性(函数也是对象的一种，是并集关系不是父子集关系)

24.prototype属性是找自己身上有没有公有属性的。

25.constructor执行继承的父级对象。

26.\_\_proto\_\_指向哪个原型就是继承哪个原型(注意是原型，直接指向函数没用)，如

Demo.prototype.case = "kiss";

function Demo() {

haode = "buhao";

}

Person.prototype.age = 19;

function Person() {

}

var person = new Person();

person.\_\_prototo\_\_ = Demo访问不了person.haode属性

而person.\_\_proto\_\_ = Demo.prototype(一般都这样写写上面的就连prototype上的属性都不能访问)能访问person.case属性

17原型链下 01：14：24

1.call和apply作用改变this指向。

call

使用方法----方法.call(里面填的东西会时方法的this全部指向这个东西，有传实参在后面传就好了);

（接用你的功能实现我的功能）如，

function Person(name, age, sex) {

this.name = name;

this.age = age;

this.sex = sex

}

function Student(name, age, sex, tel, grade) {

Person.call(this, name, age, sex);

this.tel = tel;

this.grade = grade;

}

var student = new Student('sunny', 123, 'male', 139, 2017)

apply

使用方法-------方法.apply(this指向你的东西,后面只能传数组)

2.继承的发展史

1传统模式---原型链（过多的继承了没用的属性）

2借用构造函数（注意要new出来）

3共享原型 如,

Farther.prototype.lastName = 'Deng'

function Father(){ }

function Son(){ }

Son.prototype = Father.prototype;

共享原型继承封装成方法：(注意Target和Origin是构造函数，其他的不需要直接加就好了)（Target继承Oringin的）

function inherit(Target, Origin) {

Target.protopype = Oringin.prototype

}

需要先继承后用这个方法(注意注意)（不这样的话他是改不了原型父级的原型的）

缺点：修改自己的原型会影响父级。

4.圣杯模式

function F();

F.prototype = Father.prototype;

Son.prototype = new F();

这样修改自己的原型不会影响父级的原型

封装方法

function inherit(Target, Origine) {

function O();

O.prototype = Origin.prototype; 1 注意1和2不可以写反（先new的话已经先继承了没用）

Target.prototype = new O(); 2

Target.protopype.constuctor = Target;(使原型找指向自己而不是Father.prototype)

Target.prototype.uber = Origine.prototype;(加这一步是为了想知道自己真正指向的父级是谁)

}

用的话就可以

inherit(参数就行了);

3.闭包的私有化（小媳妇）

var inherit = (function () {

var O = function () {};

return function (Target, Origin) {

O.prototype = Origin.prototype;

Target.protype = new O();

Target.prototype.constuctor = Target;

Target.prototype.uber = Origin.prototype;

}

}());

4. 命名空间

老方法 var = obj{}

新方法 利用闭包私有化。 （留一个接口）return里面的函数。如

var name = "bcd"

var init = (function () {

var name = "abc"

function callName() {

console.log(name);

}

return function () {

callName();

}

var initDeng = (function () {

var name = 123

function callName() {

console.log(name);

}

return function () {

callName();

}

下面两个人写的三个人相同的功能不影响。

name打印出bcd

init（）打印出abc

initDeng()打印出123

5.函数的连续调用return this（函数不屑return默认的值是undefined）

6.调用属性的两种方式。

obj.属性和obj['属性']（使用obj.属性里面会隐示的转化下obj[name]）

obj['属性']应用(实现调用方法数字几就出几的名字)

var Deng = {

wife1 :{name : "xiaozhang"},

wife2 : {name : "xiaozhang"},

saywife : function (num) {

return this["wife" + num];

}

}

7.对象的枚举（遍历对象）

遍历就是挨个看一次for循环

8.for in循环

prop就选中的每个属性。

for(prop(这个可以随便取) in obj) {

console.log(obj.prop)（打印不出对象里面的属性 内部会obj['prop']）

console.log(obj[prop(取得什么就写什么)])

}

灵活应用可以写成

var prop;

for(prop in obj){}

1.hasOwnProperty(属性的字符串形式)

（判断是不是原型上的属性）

对象遍历一般都是

for(var prop in obj) {

if(obj.hasOwnProperty(prop)) {

console.log(obj[prop]);

}

}(为了过滤了父级原型上的属性)

2.in 使用形式（"属性名" in obj） （这个能判断和hasOwnProperty一样但是它会把自己的原型也算上来）

3.A instanceof B(官方给出的解释，A对象是不是B构造函数构造出来的，是返回ture,不是返回false)

A instanceof B （注意：记住看A对象原型链上有没有B的原型）如

person instanceof Object返回的是ture

9.区分[]和{}

var obj = {} （直接在控制台上打印{}不可以会报错）

1.obj.constructor打印function Object(){[native code]}

2.[] instanceof Array 打印ture

3.Object.prototype.toString.call([]); 打印出'[object array]'

3的原理改变this指向。

10.（）立即执行符号 。

，运算符他会识别后面的但注意要用（）。如，

var num = (1, 2)它会识别出2

11.（）里面是表达式 (function f() {})里面的函数变成了函数表达

12.this

1.函数的预编译过程this执向window

new test();new完之后会预编译第一时候this = Object.creat(test.prototype)

2.全局作用域this指向window

3.call/aplay可以改变this指向。

4.obj.fun(); fun()里面的this指向obj(谁调用这个方法this指向谁)

var obj = {

a : function () {

console.log(this.name)

},

name : "abc"

}

obj.a(); 这个this指向obj

this笔试题------this-practice.

13.arguments.callee指向函数自身引用。如

var num = (function (n) {

if(n == 1){

return 1;

}

return n\*arguments.callee(n-1) 因为匿名函数找不到名 而用函数需要写函数名来调用，所以只有写callee

}(20))

14.func.caller在哪个环境运行就指代哪个（没多大用）

15.函数（）（） 第一个括号函数反回，第二个括号函数执行。

结束语arguments ECMAScript部分。

1.对象中的属性是function的话就是对象的方法也可以说函数。

如 var mrZhu = {

sex :male,

smoke : function() {console.log("我抽烟！！");}

}

mrZhu.smoke();打印出我抽烟了（注意调用对象属性里面的方法这样调用）

2.对象的增删改查

增 对象.东西（里面没有的东西）= "值"就行如，

mrZhu.age = "21" 则在1的例子里面加上了 age : 21,属性。

查 对象.东西（里面有的东西）就行如，

mrZhu.sex 里面就会显示male

改 对象.东西(里面有的东西) = "值"就行如，

mrZhu.smoke = "1年" 对象里面smock就会显示1年。

删 delete 对象.属性（里面有的属性）就如

delete mrZhu.sex 对象里面就没有sex属性了。

3.当一个变量没经过声名就使用会报错，但当一个属性没有使用会输出undefined。

4.一般会用this指代那个对象。

5.对象创建方法

1.var obj = {} 对象字面量/对象直接量

2.构造函数 （用来构造很多个相同但彼此独立的函数）

（1）系统自带的构造函数 new Object()

(2) 自定义

如function Person() {}

var person(可以随便取名用，这个来接收构造函数) = new Person();

3.Object.create(proto [, propertiesObject ])(proto: 对象，要继承的原型 )(propertiesObject: 对象，可选参数，为新创建的对象指定属性对象。)

6.因为构造函数和函数形式一样所以后期好维护一般构造函数都是大头峰式。

7.构造函数的内部原理

1.在函数题前面隐示的加上this = { }

2.执行this.xxx = xxx

3.隐示的返回this 如，

function Person(name, height) {

//var this = {}

this.name = name;

this.height = height;

this.say = function () {

console.log(this.say)

}

//return this;

}

注意只有new Person()之后才发生构造函数的三部//，否则就是正常函数的执行。

8.原始值undefined和null不可以有属性值 其他的原始值new一下就成对象可以有属性值成为对象。

如 var num = new Number(123); 查会查出一个123对象。

9.原始值不能有属性和方法的为什么能调用 就是经历了包装类（不能加属性但又能访问属性）

10.包装类

new String();

new Boolean();

new Number();

如

var str = "abcd";

str.length = 2;

//new String("abcd").length = 2; 不报错是因为马上会这样 但马上就会删除 delete

console.log(str)； 他打印的式abcd

如果打印的式console.log（str.length）它返回的是4，因为访问的是系统定义的new String("abcd").length

6.Math.random();随机数 返回(0, 1)的随机数

7.类数组长得像数组但又不能用数组的方法 如,

arguments是个类数组,不能用push数据进数组里面会报错.

8.成为类数组的条件

1)属性要为索引(数字)属性

2)必须要有length属性(必要条件),做好加上push属性

9.如果强行让类数组调用push方法,则会根据length属性值的位置进行扩充. 如,

var obj = {

"2" : "a",

"3" : "b",

"length" : 2,

"push" : Array.prototype.push

}

obj.push("c");

obj.push("d") 调用obj打印出obj {2 : "a", 3 : "d", length: 4}其原理是push方法封装,

Arr.prototype.push = function (target) {

obj[obj.length] = target;

obj ++;

}

1.try catch形式

try{

里面的代码发生错误，不会执行错误之后的代码且不会影响try catch之外的代码

}

catch(e（可以随便写什么。原理是系统会把错误信息封装到error对象里面把error对象传到e里面）){

他会捕捉try里面的错误代码，try没有错误时不会执行catch代码

}

2.error对象error.message和error.name

3.错误信息

4.es5.0严格模式(es3.0和es5.0冲突的部分用es5.0)注意没开启严格模式冲突部分是用3.0

启用方法 "use strict"; 在script内最上面写上字符串形式（逻辑的最顶端下面就是es5.0的模式）。

写在局部就只影响那一个写上了严格模式的函数

5.with(obj) {}改变作用域链 obj成为{}里面的代码题的最顶端的原型。es5.0不能用。

6. es2.0严格模式with，arguments.callee, func.caller不能使用，

未经声名变量不能使用（3.0是暗示全局变量），（this预编译指向window）局部this必须被赋值

(Person.call(null/undefined)赋值什么就是什么)，不能使用重复的属性和参数。

1.日期对象是系统提供好的对象。

2.查到w3c看date的属性和方法。

3.定时器(window的方法)

setInterval(function () {}, time);

每隔time时间执行一次function。 他会返回一个唯一标识可以用一个var来接受他(用来清除这个定时器clearInterval)

time只识别一次

steTimeout(function() {}, time);

隔time时间才执行这个function (只执行一次)。 clearTimeout同上

设置计时器

4.脚本化css(简介操作css)读写都是在行间。

读写css属性

dom.style.prop方法 修改行间样式(如果是css外部文件或者在head里面的都读写不到，只能读写到css的行间样式)

保留字前面加css如float ---->cssFloat

符合单词小头峰式-----> backgroundColor

写入的值必须是字符串格式----->div.style.height = "100px"

查询计算样式

window.getComputedStyle(ele,null).属性 只读 ele要获取的东西可以是div（获取的是当前元素的显示值(最终显示的不管css在哪)）

如果是相对值他会自己计算好之后返回绝对的值。

5.获取滚动条位置

window.pageXOffset/pageYOffset

document.body.scrollLeft + document.documentElement.scrollLeft 写兼容IE8一下的

6.获取视图大小

window.innerWidth/winnerHeight

ducument.documentElement.clientWidth/clientHeight 标准模式下

document.body.clientWidth/clientHeight 给怪异模式下 看html上面的！那行

document.compatMode方法可以区分怪异模式和标准模式

7.查看元素几何尺寸大小（是实时的放回不了动态的尺寸）

domEle.getBoundingClientRect();------->div.getBoundingClientRect();返回值left,right,top,bottom等

left和top代表左上角的x和y的坐标，right和bottom代表右小角的x和y的坐标

IE浏览器用这个方法不会返回width和height值 可以通过right-left计算宽，bottom-top计算高。

8.查看尺寸大小(视角上的尺寸，用的多一点)

dom.offsetWidth, dom.offsetHeifht ------>div.offsetWidth

9.查看元素位置

dom.offsetLeft/offsetTop 返回无定位父级元素，返回相对文档的坐标。对于又定位的父级元素返回相对最近有定位父级的坐标

dom.offserParent返回的是最近有定位的父级

10.让滚动条滚动

window上又三个方法

scroll(),scrollTo(); 不能累加方法一模一样

scrollBy()；能累加

括号里面填x的坐标，y的坐标

1. 实现自动阅读功能

1.事件就是实现交互体验的核心

2.绑定事件的方式

ele.onxxx = function(event) {}；

--->div.onclick = function () {} 基本都兼容

一个元素同一个事件只能绑定一次

程序this指向dom元素本身

解除事件：ele.onxxx = null/false;

obj.addEventListener("type", fn, false);

---->div.addEventListener ("click", function(){}, false ) IE9下不兼容

可以一个事件绑定多个处理函数，同一个元素执行同一个函数只能执行一次

程序this指向dom元素本身

解除事件：ele.removeListener(type, fn, false)里面的全部值要是同一个，(想要解除需要用一个函数应用，不然找不到函数)

obj.attachEvent("on+ type", fn)；

--->div.attachEvent("onclick", function(){}) IE独有的

可以一个事件绑定多个处理函数，同一个元素执行同一个函数可以执行多次

程序this指向window

ele.detachEvent("on" + type, fn)和addEventListener一样全部要是同一个

3.封装兼容性的方法要背起来addEvent(elem, type, handl);

4.事件处理模型(一个对象的一个事件类型只能出现一个模型(要么冒泡要么捕获))

事件冒泡(常规模型)：

结构上(非视觉上)嵌套关系的元素，会存在事件冒泡的功能，即同一事件上，自子元素冒泡向父元素。(自底向上)

事件捕获(不常规模型)(只有谷歌浏览器实现了)：

结构上(非视觉上)嵌套关系的元素，会存在事件捕获的功能，即同一事件上，自父元素捕获至子元素(事件源元素)。(自顶向下)

5.事件触发顺序，先捕获后冒泡。

6.事件的执行顺序是先绑定先执行(测试捕获和冒泡的先后顺序需注意绑定执行顺序)。

7.focus, blur, change, submit, reset, select等事件不冒泡

8.在每一个事件处理函数中都可以传一个形参(e)即事件对象里面有很多属性其中有一个属性方法就可以取消冒泡。

9.取消事件冒泡

w3c标准event.stopPropagation();IE9以下不支持

event.cancelBubble = true; ie独有的

10.封装冒泡函数stopBubble(event);

11.默认事件----->表单提交， a标签跳转， 右键菜单等。

12.阻止默认事件

return false；(句柄的方式阻止 只能取消句柄绑定的默认事件)

以对象属性的方式注册的事件才生效

event.preventDefault(); W3C标准，IE9一下不兼容

event.returnValue = false; 兼容IE

13.封装阻止默认事件函数cancelHandler(event);(封装不下return false)

14.事件对象

event(就是function里面的形参e)非IE存在于。 IE存在于window.event--->event || window.event

15.事件源对象(事件是在哪里执行的)

event.target 火狐只有这个

event.srcElement le只有这个

chrome浏览器两个都有

16.封装事件源对象（应用事件委托）

17.鼠标事件

click(敲击事件), mousedown(), mousemove, mouseup, contextmenu(右键出现菜单事件),

mouseover(鼠标放进去出现什么), mouseout(鼠标出来出现什么), mouseenter, mouseleave

18.区分左右键只能用moseup和mousedown事件可以区分 里面的button属性。e.button ==2 右键 e.button == 0 左键

19.键盘事件

keydown, keyup, keypress

触发顺序 keydown>keypress>keyup

keydown和keypress的区别

keydown能监听所有键盘类事件且charcode值为0(fn不能，fn键是辅助键)

keypress能监听所有字符类按键charcde为ask码(能区分字母的大小写)

20.文本类事件

input事件(改变就会触发该事件), focus(聚焦触发), blur(失去焦点触发), change(状态位发生改变才触发)(聚焦改变失去焦点才触发)

21.窗体操作类事件(window上的事件)

scroll(滚动条事件), load(效率最低,等randTree加载绘制完成之后才触发)

22.div.setCapture(); div.releaseCapture();配套使用 捕获事件 不通用只有IE有不重要

1. 事件2的作业

1.DOM只能操作html和xhtml,不能操作css(不能改变css的样式表)(可以改变行间样式，间接修改)

2.是选中html要先在body里面创建之后才可以选择。

3.document代表整个文档,在html外面。

4查

.用js选择元素括号里面填的要是字符串形式（带s都是选择出一组元素）

.getElementsById();通过ID名选择出元素 不区分id的大小写

.getElementsByTagName();通过标签名选择出元素（表单元素）

.getElementsByName();通过name选择出元素

.gtElementsByClassName();ie9一下不能用 通过class民选择出元素

下面的是静态的是副本，上面的是动态的，实时的。

.querySelector("里面css怎么选这里怎么选");

.querySelectorAll("里面css怎么选这里怎么选");选择出一组

5.生成的成组的基本上都是类数组。

6.遍历节点树（注意是节点）

parentNode 父节点（最顶端是#document）；

childNodes[第几个节点，可不写]--子节点们（所有的子节点）

firstChild-第一个子节点 lastChild-最后一个子节点

nextSibling-后一个兄弟节点 previousSibing 前一个兄弟节点

7.节点类型

元素节点 -----1 属性节点-----2 文本节点-----3 注释节点----8 document节点-------9 DocumentFragment-----11

8.基于元素点的遍历（除了chidren其他的都是IE9一下不兼容的）

parentElement（顶端是html）返回单前元素父元素节点

chidern 返回当前元素的子元素节点

firstElementChild 返回第一个元素节点

lastElementChild 返回最后一个元素节点

nextElementSibLing后一个元素节点 previousElementSibLing前一个元素节点

9.节点的四个属性

nodeName 元素的名称，以大写的形式表示，只读

nodeValue 只有文本节点和注释节点属性节点，有（text,comment）可读可写

nodeType 返回节点的类型，只可读

attributes属性节点的集合

10.hasChildNodes();方法 检测有没有子节点有turn没有false。

11.obj.prototype-->事件--->node--->Document,CharacterData,Element,Attr。

Document---->HTMLDocument,XMLDocument--->document,xmldocument

12.DOM树

getElementsById定义在Document.prototype上，Element节点上不能使用

getElementByName定义在了HTMLDocument.prototype上，不能在xml document和Element上使用

getElementByTagName定义在Document.prototype上和Element上 都能使用

HTMLDocument定义了一些常用的属性，body和head分别指其标签

document.body打印<body>标签

Document.prototype上定义了documentElement属性，指代文档的根元素，HTML中指代<html>

document.documentElement打印出<html>标签

13.增（实在js手里拿着不会显示在页面） 需要显示在页面可以document.body.appendChild("标签名")

.createElement("标签名");创建元素节点（标签）

.createTextNode("");创建节点(文本)

.creatComment("")注释节点

.createDocumentFragment()；创建文案碎片节点

14.插

appendChild(可以理解成push) 页面上已经有的剪切操作 如，

div.appendChild(span); 原来div和span是兄弟结构的变成父子结构了。（）里面不需要“”

.insertBefore(a, b); 读insert a Before b (a在b前面) 要选中他的父元素 ParentNode.insertBefore(a, b);

15删

.parent.removeChild();选中父节点删除括号里面的子标签 不需要""

child.remove(); 删掉选中的节点。

16.改

parentNode.replaceChild(new（新节点）, origin（老节点）)new代替origin

17.元素的一些属性

innerHTML = (覆盖) innerHTML += (在内容后面加东西)

innertext = 文本 (覆盖) 小心有span标签

18.元素上的方法

setAttribute("(属性名)class","on(属性值)") 如 class="on" 添加行间属性

getAttribute("标签") 取出行间属性

19.需要先定义一个标签。

总结：

1. 计算的顺序是自左向右 优先级高的先计算