1. **js是什么？**

Javascript是一门：动态的 弱类型的 解释型 脚本语言（面向对象的语言）

动态：程序执行的时候才会确定数据类型

弱类型：数据类型不固定，可以随时改变；

解释型：相对编译型的来说；

编译型语言：程序在运行之前需要整体先编译（翻译）

解释型语言：程序运行的时候，不会编译，拿一行解释执行一行；

脚本：一般都是可以嵌在其它编程语言当中执行；

1. 变量

可以变化的量

程序和进程

var a = 5;

var b;

c = 10;

var a = 5,b = 10;

var a = b = 10;

带var和不带var区别？

不带var 有可能隔山打牛 预解析不会发生

// 属性的赋值优先级比变量高（连等的情况下）

  var a = {n:1};

  var b = a;

  a.x = a = {n:2};

  console.log(a.x);

  console.log(b.x);

1. 数据

3.1 分类(2大类)

\* 基本(值)类型

\* Number: 任意数值

\* String: 任意文本

\* Boolean: true/false

\* undefined: undefined

\* null: null

\* 对象(引用)类型

\* Object: 任意对象

\* Array: 一种特别的对象类型(下标/内部数据有序)

\* Function: 一种特别的对象类型(可执行)

备注：准确来说Array和Function属于一种特别的Object类型。

3.2 数据存储

基本数据存储 数据本身

对象数据存储 地址值

3.3 对象类型数据的判断

var b1 = {

b2:[2,'atguigu',console.log],

b3:function () {

alert('hello')

}

}

console.log(b1,b1.b2,b1.b3)

console.log(b1 instanceof Object,typeof b1)

console.log(b1.b2 instanceof Array,typeof b1.b2)

console.log(b1.b3 instanceof Function,typeof b1.b3)

console.log(typeof b1.b2[2])

console.log(typeof b1.b2[2]('atguigu'))

3.4 undefined与null的区别?

\* undefined代表定义变量没有赋值 var a;

\* null代表赋值了, 只是值为null var a = null;

3.5 什么时候给变量赋值为null呢?

对象初始化 var obj = null;

删除对象 obj = null ==》 obj变为垃圾对象

3.6 关于引用变量赋值问题

1). 2个引用变量指向同一个对象, 通过一个引用变量修改对象内部数据, 另一个引用变量也看得见

var arr1 = [1,2,3];

var arr2 = arr1;

arr1[1] = 22;

console.log(arr2);

2). 2个引用变量指向同一个对象,让一个引用变量指向另一个对象, 另一个引用变量还是指向原来的对象

var obj1 = {};

var obj2 = obj1;

obj2.name = '我爱你';

obj2 = {};

console.log(obj1);

3.7 关于函数传参传值的问题

传基本数据值

var a = 10;

var b = 20;

function add(a,b){

a = 20;

return a + b;

}

add(a,b);

console.log(a);

传引用数据值

var arr = [1,2,3];

function fn(arr){

arr[1] = 22;

}

fn(arr);

console.log(arr);

1. 对象

4.1对象是什么

\* 代表现实中的某个事物, 是该事物在编程中的抽象(无序的键值对的集合)

\* 多个数据的集合体

\* 用于保存多个数据的容器

var obj = {

name:'张三'，

age:18,

eat:funcition(){

console.log('吃货')；

}

}

console.log( typeof obj.eat);

console.log( typeof obj.eat());

4.2对象的组成

\* 属性

\* 代表现实事物的状态数据

\* 由属性名和属性值组成

\* 属性名都是字符串类型, 属性值是任意类型

\* 方法

\* 代表现实事物的行为数据

\* 备注：方法是一种特别的属性==>属性值是函数

4.3对象的创建

4.4 对象操作

\* 对象.属性名: 编码简单, 但有时不能用

\* 对象['属性名']: 编码麻烦, 但通用

\* 对象[变量名]:变量里面的值会被转化成字符串，作为属性名

问题: 什么时候必须使用['属性名']的方式?

\* 属性名不是合法的标识符

\* 属性名是变量的值

4.5案例:

var a = {};

var obj1 = {m:2}

var obj2 = {n:2}

var obj3 = function(){};

a[obj1] = 4

a[obj2] = 5

a.name ='kobe'

a[obj3] = 6;

console.log(a[obj1])

console.log(a)

1. 函数

1. 什么是函数?

\* 具有特定功能的n条语句的封装体

\* 只有函数是可执行的, 其它类型的数据是不可执行的

\* 函数也是对象

2. 为什么要用函数?

\* 提高代码复用

\* 把一个项目模块化

3. 如何定义函数?

\* 函数声明（字面量）

\* 函数表达式

\* 构造函数

4. 函数的2种角色

\* 函数: 通过()使用 ==> 就称之为函数

\* 对象: 通过.使用 ==> 称之为: 函数对象

5. 回调函数

什么函数才是回调函数?

\* 你定义的

\* 你没有直接调用

\* 但最终它执行了(在特定条件或时刻)

常见的回调函数?

\* DOM事件函数

\* 定时器函数

\* ajax回调函数(后面学)

\* 生命周期回调函数(后面学)

案例

for(var i = 0; i < 3; i++){

setTimeout(function(){

console.log(i);

},0)

}

6. IIFE

理解

\* 全称: Immediately-Invoked Function Expression 立即调用函数表达式

\* 别名: 匿名函数自调用

作用

\* 不污染外部命名空间

\* 项目初始化

7. 递归

什么是递归函数:一个函数体内部调用自己

合理使用递归条件：1.有明显结束条件

1. 有趋近于结束条件的趋势

案例:

var arr = [1,2,3]

    var num = 0

    function fn(arr){

      if(arr.length==0){

        return num

      }

      num+=arr[0]

      arr.shift()

      return fn(arr)

    }

    console.log(fn(arr))

1. arguments :

1、函数可以没有形参 2、可以让函数功能丰富 3、arguments.callee

使用:兼容性封装获取文本内容

1. this是什么?

\* 一个关键字, 一个内置的引用变量

\* 在函数中都可以直接使用this

\* this代表调用函数的当前对象（函数的调用者）

\* 在定义函数时, this还没有确定, 只有在执行时才动态确定(绑定)的

如何确定this的值?

\* test() window

\* obj.test() obj

\* new test() 实例化对象

\* 事件的回调函数种指向事件源

\* test.call/apply(obj) obj

总结: 函数的调用方式决定了this是谁

案例：

基础案例(5种)

1.

Array.prototype.forEach=function(){

       console.log(this)

     };

     [1,2,3].forEach()

     var obj ={

       arr:[4,5,6]

     }

     obj.arr.forEach()

2.

var obj = {

       name:'张三',

       fn:function(){

        console.log(this)

       }

     }

     var newFn = obj.fn.bind({a:1}).bind({b:2}).bind({c:3})

     newFn()

3.

var obj = {

       name:'张三',

       obj1:{

         name1:'李四',

         fn:function(){

           console.log(this)

         }

       }

     }

     obj.obj1.fn()

4.

var name = "The Window";

    var object = {

      name: "My Object",

      getNameFunc: function () {

        return function () {

          return this.name;

        };

      }

    };

    console.log(object.getNameFunc()());

5.

var name2 = "The Window";

      var object2 = {

          name2: "My Object",

          getNameFunc: function () {

              var that = this;

              return function () {

                return that.name2;

              };

          }

      };

    console.log(object2.getNameFunc()());

6.

var p1 = {

      name: '小明',

      age: '12',

      action: function (where, doing) {

        console.log(this.age + '岁的' + this.name + '在' + where + doing);

      }

    }

    var p2 = {

      name: '小红',

      age: '15'

    }

    console.log(p1.action.call(p2, '操场上', '运动'))

1. 构造函数

什么是：new调用的函数通常叫做构造函数

作用：模拟类

构造函数特点：

    1.this指向实例化对象 内部代码相当于给实例化对象身上增加属性（实例化过程）

    2.return

           2.1 写了return关键字后面没值或者没写return或者return了一个基本数据类型 返回值都是实例化对象

           2.2 如果return了一个引用数据类型的数据 返回值就是这个引用类型数据了 不再是实例化对象

new做了那些事情

    1.在堆当中开辟了一个空间(为你的实例化对象准备的)

    2.会让函数当中的this指向于你刚才开辟的空间

    3.老老实实执行函数内部的代码 把相应的属性或者方法加到实例化对象身上(实例化过程)

    4.把实例化对象的空间地址返回出来

6.执行环境相关

1.执行上下文（执行上下文环境）：

程序在解析和运行的时候所依赖和使用的环境；

全局执行上下文环境 和 函数执行上下文环境

2.执行上下文栈：

程序为了管理执行上下文（确保程序的执行顺序）所创建的一个栈数据结构，被称作执行上下文栈；

3.预解析（变量提升）：

先解析函数：函数重名覆盖

再解析变量：变量重名忽略

var x = 10;

function fn() {

console.log(x);

}

function show(f) {

var x = 20;

f();

}

show(fn);

var a;

function a() {}

console.log(typeof a)

var c = 1;

function c(c) {

console.log(c)

c = 3

}

c(2)

var obj = {

fn2: function () {

console.log(fn2)//

}

}

obj.fn2()

7. 原型对象

    作用：节省内存 资源共享

    显示原型对象  prototype

    在哪里；只要是函数 那么函数对象身上就会有prototype属性

    隐式原型对象 \_\_proto\_\_

在哪里：只要是对象 对象身上就会有\_\_proto\_\_  [[Prototype]]谷歌浏览器显示的是这样 本质\_\_proto\_\_

8. 原型链:

是什么:查找对象属性的过程

    原型链作用:让咱们去查找对象身上没有的属性

    原型链比较重要的一些话

    1.只要是函数就会有pototype属性(显示原型对象)

    2.只要是对象 就会有\_\_proto\_\_（隐式原型对象）

    3.prototype这个对象身上一定会有一个constructor属性(构造器),他一定会指向prototype属性所在的函数

    4.所有的实例化对象隐式原型会指向其构造函数的显示原型

5.所有的原型对象都是Object的实例

案例题

function Foo () {

      getName = function () { console.log(1); };

      return this;

    }

    Foo.getName = function () { console.log(2); };

    Foo.prototype.getName = function () { console.log(3); };

    var getName = function () { console.log(4); };

    function getName () { console.log(5); }

    //请写出以下输出结果：

    Foo.getName();

    getName();

    Foo().getName();

    getName();

    new Foo.getName();

    new Foo().getName();