# JS高级笔记

面向过程和面向对象都是编程的思想,方式不一样

面向过程:凡事都是亲力亲为,所有的代码都要自己写,每一步都要很清楚,注重的是过程

面向对象:执行者成为指挥者,只要找对象,然后让对象做相关的事情,注重的是结果

面向对象的特性:封装,继承,多态

封装；就是代码的封装,把一些特征和行为封装在对象中.

面向对象的编程思想:根据需求,抽象出相关的对象,总结对象的特征和行为,把特征变成属性，行为变成方法,然后定义(js)构造函数,实例化对象,通过对象调用属性和方法,完成相应的需求.---编程的思想

对象:具体特指的某个事物,有特征(属性)和行为(方法)，对象可以看成是一坨无序属性的集合

如何创建对象?

通过调用new Object(),还有{},自定义构造函数

创建对象的方式

1. 调用系统Object()----->创建出来的对象都是Object类型的,不能很明确的指出这个对象是属于什么类型

2. 字面量的方式{}----->只能创建一个对象(一次只能创建一个)

3.工厂模式创建对象----->----->推论---->自定义构造函数的方式

自定义构造函数(优化后的工厂模式)

自定义构造函数创建对象:4件事

1.在内存中申请一块空闲的空间,存储创建的对象

2.this就是当前实例化的对象

3.设置对象中的属性和方法(为对象添加属性和方法,为属性和方法赋值)

4.把创建后的对象返回

都是需要通过new的方式

什么是原型?

构造函数中有一个属性prototype,是原型,程序员使用的

实例对象中有一个属性\_\_proto\_\_,是原型,浏览器使用的,不是很标准的,

实例对象中的\_\_proto\_\_指向的就是该实例对象中的构造函数中的prototype

构造函数中的prototype里面的属性或者方法,可以直接通过实例对象调用

正常的写法:实例对象.\_\_proto\_\_才能访问到构造函数中的prototype中的属性或者方法

per.\_\_proto\_\_.eat();//\_\_proto\_\_不是标准的属性

per.eat();

原型就是属性,而这个属性也是一个对象

Person.prototype--->是属性

Person.prototype.属性或者Person.ptototype.方法()

本身在构造函数中定义的属性和方法,当实例化对象的时候,实例对象中的属性和方法都是在自己的空间中存在的,如果是多个对象。这些属性和方法都会在单独的空间中存在,浪费内存空间,所以,为了数据共享,把想要节省空间的属性或者方法写在原型对象中,达到了数据共享，实现了节点内存空间

原型的作用之一:数据共享,节省内存空间

原型的写法:

构造函数.prototype.属性=值

构造函数.prototype.方法=值---->函数.prototype,函数也是对象,所以,里面也有\_\_proto\_\_

实例对象.prototype-------->实例对象中没有这个属性,只有\_\_proto\_\_(暂时的)

简单的原型的写法

缺陷:--->新的知识点---->原型直接指向{}---->就是一个对象,没有构造器

构造函数.prototype={

切记:如果这这种写法,要把构造器加上

};

通过原型为内置对象添加原型的属性或者方法----->原因:

系统的内置对象的属性和方法可能不满足现在需求,所以,可以通过原型的方式加入属性或者方法,为了方便开发

为内置对象的原型中添加属性和方法,那么这个内置对象的实例对象就可以直接使用了

String.prototype.方法=匿名函数;

var str="哈哈";

str.方法();---->实例对象可以直接调用原型中的属性或者方法

原型:

每个实例对象中都有一个属性\_\_proto\_\_,是原型,浏览器使用的,不标准的属性

每个构造函数中都有一个属性prototype,是原型,程序员使用的,

面向对象和面向过程都是编程思想

面向对象注重的是结果,面向过程注重的是过程

面向对象的特性:封装,继承,多态

继承:

1.通过原型实现继承,改变原型的指向,属性在初始化的时候就已经固定了,如果是多个对象实例化,那么每个实例对象的属性的值在初始化的时候都是一样的

2.借用构造函数继承,不能继承方法

3.组合继承,可以解决属性和方法的继承的问题

4.拷贝继承,就是把一个对象中的原型中的所有的属性和方法复制一份给另一个对象

创建对象的三种方式

1. 字面量的方式

2. 调用系统的构造函数

3. 自定义构造函数

原型链:实例对象和原型对象之间的关系,主要是通过\_\_proto\_\_和prototype来联系

原型的指向是可以改变,所以,js中是通过改变原型来实现继承

原型的作用:

实现数据共享,继承, 都是为了节省内存空间

如果属性和方法都需要共享,那么就把属性和方法添加到原型中

函数中的this的指向

普通的函数中this是window

构造函数中的this,构造函数一般都是创建实例对象使用的,是通过new关键字,构造函数也是函数

构造函数中的this是实例对象

方法中的this是实例对象

原型中的方法中的this是实例对象

定时器中的this是window

函数是对象,构造函数也是函数,也是对象

对象是不是函数呢?不一定

对象中有\_\_proto\_\_

函数中有prototype

Math是对象,但不是函数

apply和call方法的使用和区别

都可以改变this指向的

使用方式:

函数名.apply(对象,[参数1,参数2,参数3,...]);

函数名.call(对象,参数1,参数2,参数3,....);

我想要使用其他对象的某个方法,其他对象.方法名.apply(当前对象,参数,...);

这个方法就会被当前的对象所使用,同时这个方法中的this就是当前的对象,在调用方法的时候改变了this的指向

bind方法的使用和区别

复制一个方法或者是函数,是在复制的同时改变了this的指向

使用方式:

函数名.bind(对象,参数1,参数2，...);------>返回值是复制之后的这个函数

高阶函数---->函数的使用的方式:函数作为参数使用，函数作为返回值使用

函数作为参数使用的时候,这个函数可以是匿名函数,也可以是命名函数

作用域和作用域链及预解析

闭包:函数中有另一个函数,或者是一个函数中有另一个对象,里面的函数或者是对象都可以使用外面函数中定义的变量或者数据,此时形成闭包

function f1(num){

return function(){

console.log(num);

}

}

f1(20);

闭包模式:函数模式闭包,对象模式的闭包

闭包的作用:缓存数据,延长作用域链,同时也是缺点,函数中的变量不能及时的释放

沙箱:就是一个环境,也可以叫黑盒,在这个环境中模拟外面真实的开发环境,完成需求,效果和外面的真实的开发环境是一样的

沙箱避免命名冲突

递归:函数中调用函数自己,递归一定要有结束的条件,否则就是死循环

递归:一般应用在遍历上

递归轻易不要用,效率很低,