阻止js死循环网页卡住的方法-慕课？公众号？

15.作用域和内存

作用域：按照规则对函数和变量进行存储。

作用域链：对函数、变量访问。

17.数组的去重

18.时间对象

时间分为GMP和UTL时间。

19.函数的创建

1.函数声明

2.函数表达式

函数生成器返回的结果是一个对象，一个为value，一个是done，done在前几次运行时返回的是false，最后返回的是true。

20.JS中的作用域

立即执行(自执行函数)INFE ?

立即执行函数的作用：

1避免作用域污染

2.提升性能

3.有利于压缩

4.解决全局命名冲突：实参传window.jQquery;前边有人定义undefined=true在立即执行函数中形参写为undefined实参不传

5.保存闭包的状态

6.使用UMD规范颠倒代码的运行顺序

建议：在立即执行函数的()前面加上分号，因为在压缩过程中可能和别的代码压缩到一行，混乱报错。

JS作用域缺陷：不提供块级作用域；使用立即执行函数模拟块级作用域。

ES3中一般情况没有块级作用域，但是有三种方法通过欺骗词法分析来创建和修改作用域：

1.with{} : with后的{}是块级作用域

2.try{}catch{} : catch后的{}是块级作用域。将要声明的变量放在try中，以异常的形式抛出，然后再catch中接收变量，即变成块级作用域中的变量。

缺点：命名多个块级作用域变量时使用多个try catch嵌套语句；try catch性能没有正常代码执行快，但是没有技术层面理论依据解释其性能低。

3.eval() 是函数，创建作用域

ES6使用let定义块级作用域下的变量，但是不能将所有的var使用let代替：因为let声明的变量属于新的作用域，而不是在当前作用域下，当变量与变量之间存在隐式依赖时，可能回出现问题？？

21.任意门详解闭包

在js中函数本身也是变量，所以可以将函数作为值来应用。(创建函数表达式时；将函数当成对象的属性值；把函数当成数组中的数据项)

将函数作为另外一个函数的参数使用；将函数作为返回值使用

1.将函数作为参数使用

操作函数：

22.包装对象和内置对象

23.面向对象设计模式

(1)工厂模式

(2)原型模式：所有的属性都是共享的。

(3)构造函数模式

(4)组合模式

所有函数上都有一个特殊的属性prototype，其指向一个特殊的对象称为原型对象。圆形对象上只有一个属性constructor，又指向了这个构造函数。

所有的对象上都有一个隐藏的属性\_\_proto\_\_，指向了当前对象的构造函数所对象的原型对象。

由\_\_proto\_\_连接起来的对象和原型对象为原型链，原型链的最后会指向null。

访问对象的属性会顺着原型链访问，但是当设置属性的时候只会在当前对象上设置属性。

24.JS中的继承

JS是面向过程的语言。

模拟继承

24.1原型链继承

有两个函数A和B，使B的实例继承A实例的属性和方法。

思想：让B的原型对象指向A的实例。

B.prototype = new A();

优点：(1)使用简单；

(2)在B.prototype上继续增加属性和方法后，B的子类都可以访问到。

(使用B.prototype=A.prototype也可以做到上一点，但是修改B时会破坏A的原型对象的属性和方法。)

缺点：(1)当为子类增加属性和方法时，增加的过程必须加在B.prototype=new A()之后，否则增加的属性和方法会被覆盖掉；

(2)无法实现多继承

(3)A的原型对象上所有的属性都是共享的(针对父类实例值类型成员的更改不影响，针对父类实例引用类型成员的更改会影响其他子类实例)。当对原型对象的一些属性操作时，会影响所有的实例(可以将不允许改变的如属性放在构造函数中，原型对象上一般存放方法)

(4)没办法传递参数

24.2构造函数继承

思想：在子类的构造函数中使用call或apply将父类的构造函数再运行一遍。

Function B(){ A.appy(this,[arguments])};//apply传递参数数组，call传递参数序列。

优点：(1)可以实现多继承；

(2)解决了A原型对象的共享问题；每次new B都是重新执行了一遍A，相当于创建了一个新的对象。

(3)可以传参；

缺点：(1)new B创建出来的实例只是子类的实例，不是父类的实例，B.prototype\_\_proto\_\_指向的是Object的原型对象，不是A的原型对象。

(2)借用构造函数模式只是将父类运行一遍，而没有涉及到父类的原型对象A.prototype。只能继承构造函数内的属性和方法，不能继承原型的属性和方法。

(3)所有的属性都是在构造函数中运行的，没法进行复用。且占用内存。

24.3组合式继承(伪经典式继承)

思想：在子类构造函数中使用call或apply运行父类构造函数，再让子类的原型指向父类的实例。

A.apply(this,[arguments])

B.prototype=new A()

B.prototype.constrctor=B;

缺点：(1)父类的构造函数被调用了两次。

(2)当使用new B创建实例时，在实例和原型对象上都有name属性，同名覆盖，子类实例将子类原型上的同名属性屏蔽了。(在构造函数中设置的属性和方法在实例和原型对象上都存在一份)占用多余内存。

24.4原型式继承

两个对象为字面量对象A和B。使B继承A

(B.\_\_proto\_\_=A，；老版本浏览器没有\_\_proto\_\_属性)

Function f(){}

f.prototype = A;

B = new f();

封装function aa(){

Function f(){};

f.prototype=A;

return new f();

}

B = aa();

书中将这种方法封装为object();

这种方法也是ES5中的Object.create(arg1,arg2)方法：参数arg1是要继承的对象，参数arg2可以配置属性上的状态，可写可枚举等。。

缺点：看起来不像继承，为了看起来像继承->寄生式继承

24.5寄生式继承

同原型式继承，让一个对象继承另一个对象，而不是构造函数。

使用函数将原型式继承包裹起来，在包裹的过程中，又会创建新的对象增加一些属性和方法-对象增强。看书？？

24.6组合寄生式继承(完美继承)

解决组合模式的两个缺点，修改B.prototype=new A()这行代码，改为

B.prototype.\_\_proto\_\_=A.prototype。

创建P对象继承A.prototype,让P.constructor=B,B.prototype=P。

P = Object.create(A.prototype)

P.constructor=B;

B.prototype=P;

同B.prototype=Object.create(A.prototype);B.prototype.constructor=B;

24.7拷贝继承

继承：让一个对象拥有另外一个对象的属性和方法

可以直接将A中的属性拷贝到B中。

缺点：属性不是原始类型，而是对象的会出现问题，拷贝的是对象的引用->浅拷贝。

(深拷贝：拷贝时先判断属性值的类型，如果是对象，则再循环一次属性值的对象，递归。)

作业：查看underscore和lodash源码，使用js开发其API？？