█js数据类型

◎JS数据类型：

基本数据类型

undefined

null

number

string

boolean

复杂数据类型

object

◎对象的种类：

Js内置的（如Number）

宿主环境（如window）

自己创建的

█

对象的私有变量，以及通过get和set来访问和修改私有变量

<script>

        var p = {

            name: 'jojo',

            work: function () {

                console.log('working...')

            },

            \_age:18, //带有下划线开头的，是对象的私有变量，不能在外部直接访问（虽然可以，但不建议在外部访问）

            // 可以使用下面的get和set方法来从外面访问和修改私有变量\_age

            get age () {

                return this.\_age

            },

            set age (val) {

                if (val < 0 || val > 150) {

                    throw new Error('invalid value')

                } else {

                    this.\_age = val

                }

            }

        }

        // 访问私有变量\_age

        console.log(p.age) //18

        // 修改私有变量\_age

        p.age = 30

        console.log(p.age) //30

    </script>

█对象的访问

通常使用点和中括号来访问对象中的属性或方法，使用中括号来访问有如下两点特点：

第一

<script>

        var a = {

            1: 'haha'

        }

        // 如果用a.1来访问会报错，因为JS不建议用数字来做键名

        // console.log(a.1)

        // 使用[]来访问则可正常运行

        console.log(a[1]) //haha

    </script>

第二，使用中括号可以以变量的形式来访问

<script>

        var a = {

            b1: '11',

            b2: '22'

        }

        var b = 'b1'

        console.log(a[b]) //11

    </script>

█使用&&来访问不确定的属性

<script>

        var p = {

            np: {

                nnp: 'haha'

            }

        }

        // 当不确定p里面是否有某个属性时，可以用&&

        var r = p && p.np && p.np.nnp

        console.log(r) //haha

        var r2 = p && p.np && p.np.nnnp

        console.log(r2) //undefined

    </script>

█遍历对象

For in 来遍历对象是无序的

<script>

        var p = {

            a: 'aa',

            b: 'bb',

            c: 'cc'

        }

        for (x in p) {

            console.log(x + ':' + p[x]) //不能用p.因为x是变量所以要用中括号

        }

        // 输出：

        // a:aa

        // b:bb

        // c:cc

    </script>

如需有序遍历对象，可使用Object.keys方法

<script>

        var p = {

            a: 'aa',

            b: 'bb',

            c: 'cc'

        }

        // Object.keys方法会返回一个有序的数组

        var ap = Object.keys(p)

        console.log(ap) //["a", "b", "c"]

    </script>

█关于对象属性的一些操作

查看对象中是否有某个属性

删除属性

查看对象属性的特性

<script>

        var p = {

            a: 'aa',

            b: 'bb',

            c: 'cc'

        }

        console.log('a' in p) //true

        console.log(p.hasOwnProperty('a')) //true

        console.log(p.b) //bb

        delete p.b //删除P的b属性

        console.log(p.b) //undefined

        console.log(Object.getOwnPropertyDescriptor(p, 'c')) //{value: "cc", writable: true, enumerable: true, configurable: true}

    </script>

█构造器或者类的研究

<script>

        // 构造器就是构造函数或者也可以理解为类

        // 使用Object构造器生成实例a和a2，通过constructor查询它们的构造器都是Object()

        var a = {}

        console.log(a.constructor) //ƒ Object() { [native code] }

        var a2 = new Object()

        console.log(a2.constructor) //ƒ Object() { [native code] }

        // 以下会报错，因为数字1没有构造器

        //console.log(1.constructor) //Uncaught SyntaxError: Invalid or unexpected token

        // 使用Number构造器创建实例b和b2，通过constructor查询它们的构造器都是Number()

        var b = 1 //把数字1赋值给变量，变量是对象，所以有构造器

        console.log(b.constructor) //ƒ Number() { [native code] }

        var b2 = new Number(1)

        console.log(b2.constructor) //ƒ Number() { [native code] }

        // 那么Object()和Number()又是谁创建出来的呢？通过constructor查询，它们是Function()创建出来的

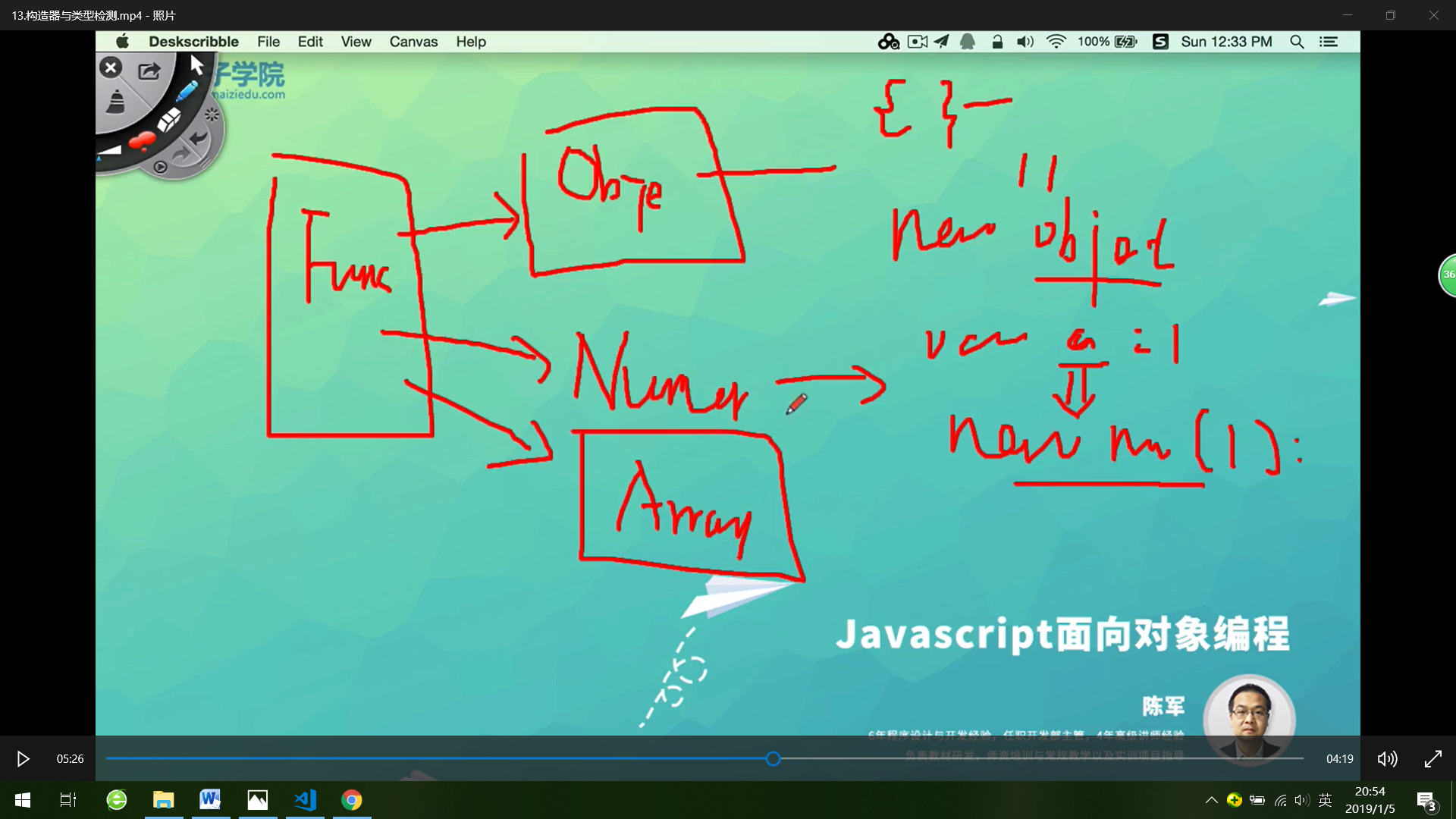
        console.log(Object.constructor) //ƒ Function() { [native code] }

        console.log(Number.constructor) //ƒ Function() { [native code] }

        // 那么Function()是谁创建的呢？通过constructor查询，它是自己创作的自己，所以它是最顶端的构造器，它创作了一切

        console.log(Function.constructor) //ƒ Function() { [native code] }

    </script>



█工厂模式

<script type="text/javascript">

            function createPerson(name, age, job){

                var o=new Object();

                o.name=name;

                o.age=age;

                o.job=job;

                o.sayName=function(){

                    alert(this.name);

                };

                return o;

            }

            var person1=createPerson("Tom", 29, "worker");

            var person2=createPerson("Jerry", 22, "teacher");

            console.log(person1.name); //Tom

            person2.sayName(); //Jerry

        </script>

以上例子还可以简写，直接return一个{} ，省略了在构造函数中创建一个对象o并return o

<script type="text/javascript">

            function createPerson(name, age, job){

                // 直接return一个{}

                return {

                    name: name,

                    age: age,

                    job: job,

                    sayName: function () {

                        alert(this.name)

                    }

                }

            }

            var person1=createPerson("Tom", 29, "worker");

            var person2=createPerson("Jerry", 22, "teacher");

            console.log(person1.name); //Tom

            person2.sayName(); //Jerry

        </script>

█构造函数模式

<script type="text/javascript">

            function Person(name, age, job){

                this.name=name;

                this.age=age;

                this.job=job;

                this.sayName=function(){

                    alert(this.name);

                };

            }

            var person1=new Person("Tom", 29, "worker");

            var person2=new Person("Jerry", 22, "teacher");

            Person("Alice", 25, "stundent"); //不使用new直接调用函数，那么this指的上一级对象，这里添加到了window

            console.log(person1.age); //29

            person2.sayName(); //Jerry

            window.sayName(); //Alice

        </script>

◎函数的调用方法有两种：

一种是直接调用方法，即，函数名加括号

另一种是构造函数调用方法，即，new加空格加函数名加括号

◎当在全局作用域中调用一个函数时，this对象总是指向Global对象（在浏览器中时window对象）。因此，上面例子中Person("Alice", 25, "stundent")将属性和方法添加给了window对象。之后可以通过window对象来调用其属性或方法。例如window.sayName

◎使用new操作符调用函数的方法叫构造函数方法。以这种方法调用的构造函数会经历以下4个步骤：

1. 创建一个新对象。
2. 将构造函数的作用域赋给新对象（因此this就指向了这个新对象）。
3. 执行构造函数中的代码（为这个新对象添加属性和方法）。
4. 返回新对象。

◎操作习惯：当创建一个函数时，如果希望创建构造函数，则函数名的首字母建议大写，这样以后便于区分哪些是普通函数，哪些是构造函数。