预习作业

- JavaScript语法
- JavaScript对象

知识要点

- JavaScript语法
- 函数
- 内置对象
- BOM对象
- DOM对象
- 自定义对象
- 闭包
- 原型链

授课笔记

- 概念
  - ECMAscript(ES6)：根据MCMA-262标准，实现的通用脚本语言规范。
  - JavaScript(JS)：通用的跨平台脚本语言，他遵守ECMA262的标准，换句话说就是ECMAscript的方言，其他的还有VBScript、微软的jscript等。
  - TS：TypeScript，对JavaScript进行封装的框架，主要进行类型约束，需要编程成JS后才能使用。
  - 一个完整的JavaScript由以下三个不同的部分组成，它们分别是：
    1. 核心语法（ECMAScript）
    2. 浏览器对象模型（BOM）
    3. 文档对象模型（DOM）
- 功能：
  - 动态改变页面元素
  - 动画
  - 表单验证
  - Ajax [XHR/fetch]
- JavaScript使用：
  - 外部脚本：`<script src="*.js"></script>`
  - 内部脚本：`<script></script>`
  - 内联脚本：`<div on* = "">` **(禁用)**
- JavaScript语法
  - **JavaScript是一种弱类型语言。**
  - 标识符：和Java一样。
  - 数据类型：
    - number：数字类型

包含了整数和实数，NaN（所谓NaN，英语全称Not a number，表示不是一个数。如果任何一个数和NaN进行操作的话，返回的会是NaN）NaN与任何值都不相等，包括它自己本身！！！

- string：字符串类型
- boolean：布尔类型
- Object：对象，null表示一个空的对象
- undefined：定义了一个变量但是没有被赋值（**重点**）
- function：函数。（**重点**）

```
var temp; //int i;  String s;
var num = 1;
var name = "david";
var flag = true;
var arr = [1,2,3];
var foo = function (){};
console.log(typeof temp);
console.log(typeof num);
console.log(typeof name);
console.log(typeof flag);
console.log(typeof arr);
console.log(typeof foo);
```

- 变量和常量：
  - 因为弱类型，变量可以不声明直接使用。**（但是不建议）**
  - var和let
    1. var 和 let 第一点不同就是 let 是块作用域，即其在整个大括号 {} 之内可见。
    2. 在变量声明之前就访问变量的话，会直接提示 ReferenceError，而不像 var 那样使用默认值 undefined。

```
<Script>
        function foo() {  //public int foo (int a ,int b){} => public
foo(a,b)
            console.log(sum);
            let sum;
            for (var i = 0; i < 10; i++) {
                sum += i;
                var name = "david";
            }
            console.log(i);
            console.log(name);
        }
        foo();
</Script>
```

  - const 表示常量（Java 是？）

- 运算符：===和==的区别？前者类型和值都必须相等，后者值相等。`“1” == 1`
- 流程控制：for-in & for-of

```
<script type="text/javascript">
  var x
  var mycars = new Array()
  mycars[0] = "Saab"
  mycars[1] = "Volvo"
  mycars[2] = "BMW"
  for (x in mycars){
    document.write(mycars[x] + "<br />")
  }
</script>
```

- 函数
  - function函数。弱类型所以函数声明过程中涉及的变量类型都可以省略。（参数类型、返回值类型）
  - 函数调用和Java不同，不需要参数匹配。
  - **函数也是类型**
  - 函数创建的两种方式（区别在于函数提升不同）
    - 函数声明

```
function functionName(arg0,arg1){
    //do some operation
}
functionName();
```

    - 函数表达式
      - 匿名函数表达式(一般使用)

```
var functionName = function(arg0,arg1){
    //do some operation
}
functionName();
```

      - 命名函数表达式（特殊使用）

```
var functionName = function foo(arg0,arg1){
    //do some operation
}
functionName();
```

      [命名函数表达式的意义]

  - 自执行函数
    - 传统声明函数(Java匿名内部类)

```
(function () {
    //do some operation
}()); // 推荐使用这个
(function () {
    //do some operation
})(); // 但是这个也是可以用的
```

- 变量声明函数

```
var foo = function(){
    //do some operation
}()
```

- 内置对象【基本引用类型】
  - 数组（Array）
    - 数组声明因为弱类型，统一使用Array。
    - 数组声明可以不设置长度。
    - 数组下标可以使用字符串标识符。

```
<Script>
  var names = new Array();//写长度，数组
  names['one'] = "david";//Map
  names[2] = "tom";//list
  console.log(names['one']);
  var nums = [1,2,3,4,5];
</Script>
```

  - 原始包装对象
    - 原始包装类型有三个，分别是String、Boolean、Number。
    - 同Java一样，都是可以通过new关键字实例化的对象。
    - String -> number : parseInt()， parseFloat() 【ES6新版本建议使用Number对象调用该方法】
    - 常用方法包括：split()、replace()、indexof()、substring()、toString()等。
  - 日期
    - Date对象表示日期。同Java一样可以使用getter方法获得日期的各个部分。
    - JavaScript没有模板，需要特殊格式日期，使用getter方法拼接。“2020-10-01“

```
var today = new Date();//可传参数
console.log(today);
console.log(today.toLocaleString());
```

  - RegExp对象：正则表达式使用对象。let reg = /^\w{6,20}$/　reg.test("abdc")
  - Global对象：全局对象，不需要显示实例化，直接使用。全局属性和函数
  - Math对象：数学函数对象。
- BOM对象：浏览器对象模型(**window** 可以省略)

    - 属性
        - document(DOM) = HTML `<html>`
        - location.href="" URL地址栏地址 **赋值->页面跳转**
        - history.go(-1) 历史,正数向前（forward），负数向后（back）
    - 方法
        - setTimeout("",1000) : 过指定时间，执行一次指定的代码。
        - setInterval("",1000)：每过指定时间，执行一次指定的代码。
        - var t = setTimeout(); cleanTimeout(t);
    - 事件 -> 事件驱动开发 (事件触发函数)
        - onload
- DOM对象：文档对象模型（core DOM | HTMLDOM | XMLDOM）
    - document ： document.getElement*** 获得需要操作的节点。$("#div")
    - element：element对象获得其他element或者操作属性和事件。node
    - attribute：element的属性，可以通过element.attribut()方式调用。
    - event：Event 对象代表事件的状态，比如事件在其中发生的元素、键盘按键的状态、鼠标的位置、鼠标按钮的状态。通常和方法一起使用，是方法的参数。
    - 事件驱动思路：**触发对象 -> 触发事件 -> 定义函数 -> 操作对象**
- 自定义对象
    - 标准

```
var person = new Object();
person.name = "lisi";
person.age = 21;
person.family = ["lida","lier","wangwu"];
person.say = function(){
  alert(this.name);
}
```

    - **字面式(JSON)** {"name":"lisi","age":21} JSON.

```
var person = {
  name: "lisi",
  age: 21,
  family: ["lida","lier","wangwu"],
  say: function(){
    alert(this.name);
  }
};
```

    - 工厂式

```
function personFactory(name,age,family) {
    var o = new Object();
    o.name = name;
    o.age = age;
    o.family = family;
    o.say = function(){
        alert(this.name);
    }
    return o;
}
var person1 = personFactory("lisi",21,["lida","lier","wangwu"]);
console.log(person1 instanceof Object); //true
console.log(person1 instanceof Person); //false
```

注意：instanceof无法判断它是谁的实例，只能判断他是对象。

- 构造函数式

```
function Person(name,age,family) {
    this.name = name;
    this.age = age;
    this.family = family;
    this.say = function(){
        alert(this.name);
    }
}
var person1 = new Person("lisi",21,["lida","lier","wangwu"]);
console.log(person1 instanceof Object); //true
console.log(person1 instanceof Person); //true
```

注意：instanceof即可以判断为object对象，也可以判断他是Person类型的对象。

- 闭包
  - 作用：
    - 可以在外部调用到函数内部的变量。
    - 局部变量不改变作用范围的同时，延迟变量的生命周期。保证局部变量不被垃圾回收。
  - 语法特点：
    - 局部变量（需要在函数外部访问）
    - 函数套函数
    - 内部函数占有局部变量

```
function funA(){
  var a = 10;  // funA的活动对象之中;
  return function(){   //匿名函数的活动对象;
     alert(a);
  }
}
//有return 所以需要使用变量接受return结果
var b = funA();
//return结果是一个方法。所以需要调用这个方法。方法也是对象。
b();
```

- 闭包特点：
  - 让外部访问函数内部变量成为可能；
  - 可以避免使用全局变量，防止全局变量污染；
  - 局部变量会常驻在内存中；
  - 会造成内存泄漏（有一块内存空间被长期占用，而不被释放）
  - 闭包中的this指向window
- 场景
  - 单例模式
    - 函数自执行
    - const

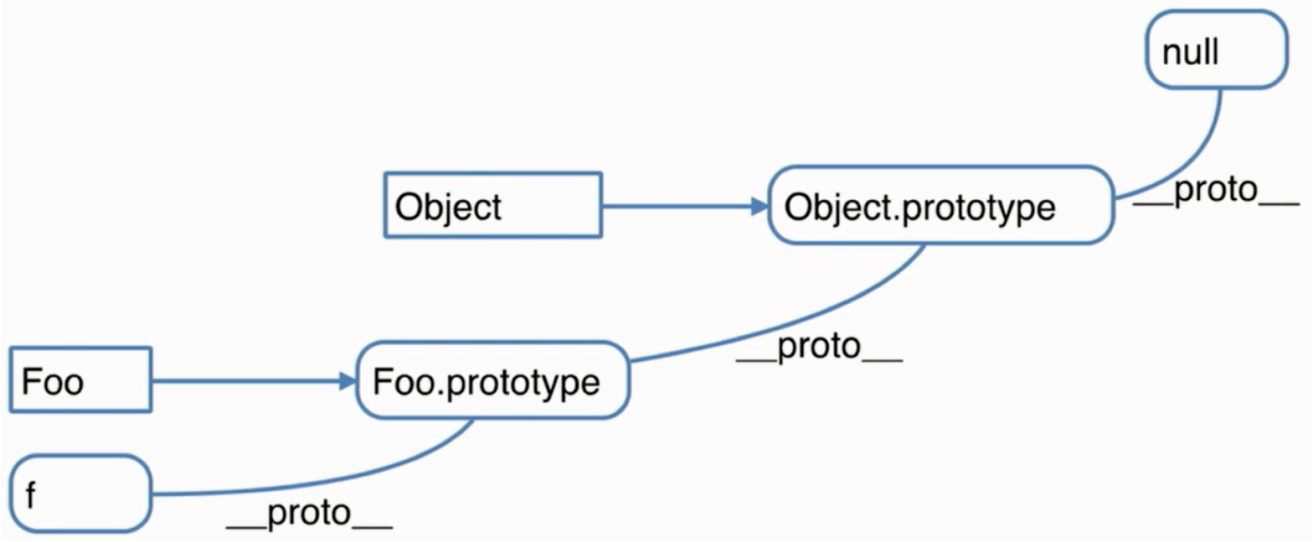
```
var Head = function () { // 匿名自执行函数
    var HeadClass = function () { }; // 声明HeadClass对象，无法在外部直接调用
    var instance; // 声明一个instance对象
    return function () {
        if (instance) { // 如果已存在 则返回instance
            return instance;
        }
        instance = new HeadClass();// 如果不存在 则new一个
        return instance;
    }
}();
var a = Head();
var b = new Head();
console.log(a===b) // true
var a = HeadClass(); // 报错,HeadClass is not defined
```

  - 封装
    - 不自执行
    - return obj{setter&getter}

```
function User () { // 声明User类
    var name = "";
    var age = 0;
    var a = {
```

```
        setName: function(str){
            name = str;
        },
        getName: function () {
            return name;
        },
        getAge: function () {
            return age;
        }
    }
    return a;
};
var user = new User();
user.setName("david");
console.log(user.getName());
console.log(user.getAge());
```

- 原型链
  - 主要用来实现继承操作。



  - prototype：每个对象（类）都有一个prototype属性，这个属性指向函数的原型对象（父类）。

```
function Person(age) {
    this.age = age
}
Person.prototype.name = 'kavin'
var person1 = new Person()
var person2 = new Person()
console.log(person1.name) //kavin
console.log(person2.name)  //kavin
```

Person：JS对象 = Java类；person1：JS实例 = Java对象

  - _proto_：这是每个对象(除null外)都会有的属性，叫做__proto__，**这个属性会指向该对象的原型。**

```
function Person() {

}
var person1 = new Person();
console.log(person1.__proto__ === Person.prototype); // true
```

- 使用闭包会导致原型链继承失效 - 示例代码

```
let father = { // object
    fatherName : "tom"
}
function Son (){ // function
    let name;
    return {
        setName:function (_name){
            name = _name;
        },
        getName: function (){
            return name;
        }
    }
}
Son.prototype = father;
let son = new Son();
console.log(son);
```

- 使用call实现继承

```
function Parent(username){
  this.username = username;
  this.showName= function(){
    alert(this.username);
  }
}
//子类Child继承父类Parent
function Child(username,password){
  //使用call方式
  Parent.call(this,username)//将parent中的this指向修改为child对象
  this.password = password;
  this.show = function(){
    alert(this.password);
  }
}

var child = new Child("zhangsan",123);
child.showName();//parent中的this是child，child继承了parent中的所有的属性和方法。
```

- 闭包+call实现继承功能综合案例

```
function Father(name){
    this.fatherName = name;
}
function Son (fname){ // function
    let name;
    Father.call(this,fname);
    let fatherName = this.fatherName;
    return {
        setName:function (_name){
            name = _name;
        },
        getName: function (){
            return name;
        },
        showFather: function (){
            return fatherName;
        }
    }
}
let son = new Son("king"); //father -> king
son.setName("tom");//son -> tom
console.log(son.showFather());
console.log(son.getName());
```

随堂练习

函数参数练习

```
<script>
    function foo(a,b,c) {
        console.log(a + b + c);
    }
    foo(1,2,3);//6
    foo(1,2);//NaN
    foo("hello","david");//hellodavidundefined
    foo();//NaN
</script>
```

字符串转数字

```
var ret = parseInt("10") + parseFloat("12.2");
console.log(ret);
console.log(typeof parseInt("10"));
console.log(typeof ret);
```

获得系统时间

```
function showTime(){
    //获得date
    var now = new Date();
    //获得数据的各个部分
    var year = now.getFullYear();
    var month = now.getMonth();//0-11
    var date = now.getDate();
    var hours = now.getHours();
    var min = now.getMinutes();
    var sec = now.getSeconds();
    //按照需求，组装数据
    var str = year + "年" + (month+1) + "月" + date +"日 " + hours +":"+min+":"+sec ;
    //展示
    document.getElementById("show").innerText = str;
}
window.onload = function(){
    setInterval(showTime,1000);
}
```

课后作业

1. 格式化输出当前日期，“2021-01-25 14:00:00”。（innerText onload）
2. 在作业1的基础上，让时间变化，同步系统时间。（setTimeout）
3. 使用setTimeout实现图片轮播。（onload）
4. 在作业3的基础上，实现鼠标悬停效果。（onmouseover / onmouseout）
5. 使用原型链描述Pet和Dog。（需求参考Java练习）。
6. 使用闭包描述Pet和Dog。