中国科学院大学
University of Chinese Academy of Sciences

个人作品

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

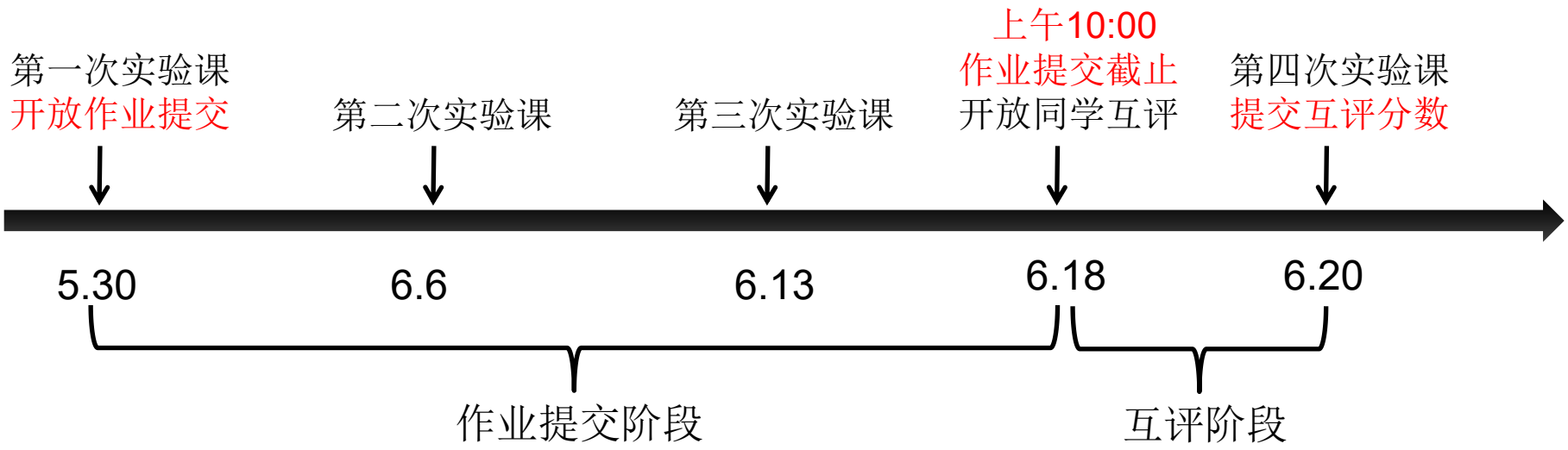
个人作品实验安排

周次	日期	实验课	课程内容	作业提交内容	截止时间
14	5月30日	制作静态网页	HTML结构 HTML语法	作品名称、包含index.html的zip文件	6月18日 10:00前
15	6月6日	制作网页版计算器	修改计算器的结果 读取操作数 执行选中的运算		
16	6月13日	制作网页版时钟	添加静态时钟 让时钟动起来 Debug方法 对比JS和Go		
17	6月20日	网页打分	小组讨论与互评	评分结果	

本课程将会以“王小二”编写一个网页博客介绍他发明了网页版时钟和计算器的伟大事迹为例，讲述相关的编程知识。

评分标准

- 6月18日 10:00 前可反复提交并预览
- 教师评分+同学互评
  - 同学互评开放时间：6月18日 17:00 - 6月20日 13:30
  - 评分维度：创造性与完成度



评分标准

- 创造性
  - 0分：无创意，完全依赖模板或已有内容，无任何创新或个性化表达。
  - 1分：创意欠缺，偶有微小改动，但整体仍以模仿为主，尝试加入少量新元素，但效果不明显或逻辑牵强。
  - 2分：具备基础创新性，结合现有方法或内容，有一定重组或改进。
  - 3分：较有创意，内容或设计具有独特性和启发性。
  - 4分：很有创意，且体现了精心的设计与实现。

评分标准

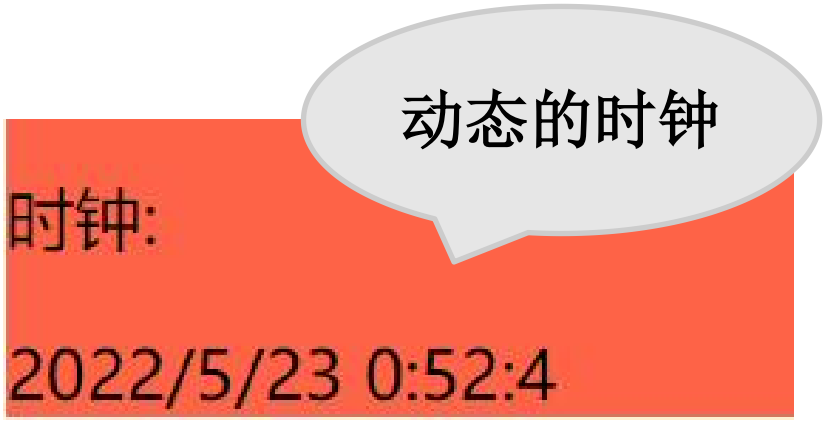
- 完成度
  - 0分：未完成，无法展示核心功能，或存在抄袭，使用他人材料无致谢。
  - 1分：完成度低，有严重缺陷，仅能展示很少部分功能，使用他人材料或代码比例过大。
  - 2分：基本完成，有bug，能展示主要功能，多处使用他人资源但致谢规范。
  - 3分：完成，仅有少量bug，几乎不影响展示功能，使用少量外部资源且致谢规范。
  - 4分：完成度高，没有bug，功能展示完整，几乎完全原创，致谢规范。

作品提交

- 时间：2025 年 6 月 18 日 10:00 前
- 提交内容：作品名称 + zip压缩包
  - 在https://part.cs101.cs.ac.cn/submission的作品提交界面提交包含index.html的zip压缩包及作品名称
  - 直接选中包括index.html的所有文件，右键压缩为zip
  - 不要将index.html放到文件夹里，压缩文件夹

我们将实现一个网页版时钟：

index.html



第三周 提纲

- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

JavaScript 编程知识—Date 对象

- 王小二写完计算器，开始写网页版时钟，并且希望把年月日、时分秒显示在时钟上，于是学习了JavaScript的Date对象：
- Date对象的**创建**

```
var date = new Date();  // 变量date中存一个Date对象
var date = new Date;    // 创建时可以不加圆括号
```

- Date对象的**方法的使用**

```
date.getDate();          // 返回该月第几天（从1开始）
date.getDay();           // 返回该周第几天（从1开始，星期日为0）
date.getFullYear();      // 返回年份
date.getMonth();         // 返回月份（从0开始）
date.getHours();         // 返回小时（在一天中）
date.getMinutes();       // 返回分钟（在一小时中）
date.getSeconds();       // 返回秒（在一分钟中）
```

添加静态时钟

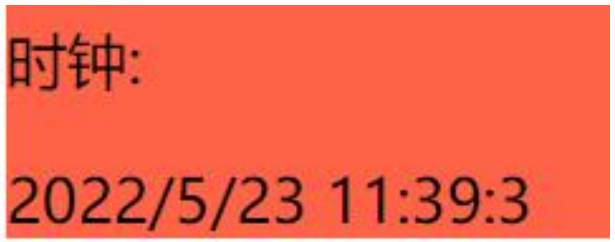
```
<div id="after">
……
Clock:
<p id="clock"></p>
</div>
<script>
  ……
  var date = new Date;
  var year = date.getFullYear();
  var month = date.getMonth() + 1;
  var day = date.getDate();
  var hour = date.getHours();
  var min = date.getMinutes();
  var second = date.getSeconds();
  var time = year + "/" + month + "/" + day + " " + hour + ":  " + min + ":
" + second;
  var clock = document.getElementById("clock");
  clock.innerHTML = time;
</script>
```

get time:year, month, day, hour, minute, second

当操作数之一的类型为字符串时，+运算符表示字符串连接（数字回先转换为字符串）

更改id属性为clock标签之间的内容

效果：



第三周 提纲

- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

JavaScript 动画

- 王小二写了一个时钟，但是这个时钟只是显示了一条日期，并不是一个动态的时钟，为了让这个时钟动起来，王小二学习了下面的函数：
- setTimeout函数
  - 用法：setTimeout(函数名，时间（单位毫秒）,参数1,参数2,…,);
  - 含义：告知浏览器，在指定的毫秒过后，执行由第一个参数指定的函数（以第三个参数以及后面的参数作为此函数的参数）
  - 仅仅告知，不需要等待浏览器执行所告知的函数，告知完继续执行后面的语句
- 使用举例：

```
var a = 0;
setTimeout(console.log,1000,a);  // 1000ms后执行
console.log(a);
```

- 把显示时间的代码放到一个函数里，每 1000ms 就调用一遍：

只需要调用一次 run_clock 函数

```
run_clock();
function run_clock(){
    …(原显示时间代码)
    setTimeout(run_clock,
1000);
}
```

每次函数执行完，都会告诉浏览器，1000ms后再调用“我”一遍

效果：

时钟:
2022/5/23 11:41:47

时钟:
2022/5/23 11:41:57

时钟:
2022/5/23 11:42:12

JavaScript 动画

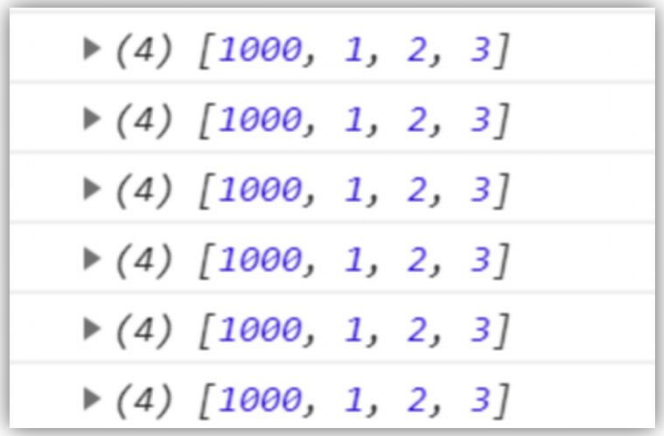
- 至此，王小二成功地完成了网页版时钟，但是王小二在使用setTimeout的时候，巧妙地避开了setTimeout的一个坑：
- 另一个使用setTimeout的例子：

```
var ar=[0,1,2,3];
for(var i=0;i<1000;i++){
        setTimeout(console.log, 1000*i, ar);
        ar[0]++;
}
```

- 控制台上会打印什么？
  - 会逐渐打印[0,1,2,3],[1,1,2,3],…吗?

JavaScript 动画

- 控制台打印：



- 当传递给函数的参数是一个对象时，实际传递的只是对象的地址，而非复制整个对象。
- setTimeout每次把数组ar的地址传给浏览器，而且由于ar只有一个，当setTimeout告知过浏览器1000次后，ar已经变成了[1000,1,2,3]。
- 即使时间达到，setTimeout告知浏览器要执行的函数必须在正常的代码执行完后再执行。
- 于是后面console.log(ar)打印的全部是[1000,1,2,3]。

JavaScript 动画

- 其它使用setTimeout的例子
  - 传递数字
    - 传递的是数字的副本，而非地址
  - 传递对象后，修改保存对象的变量
    - 传递的是原对象的地址
      - 共定义了4个对象，先后赋值给x
      - 每次setTimeout的第3个参数是不同的对象
  - 传递对象后，修改对象的属性
    - 传递的是原对象的地址
      - 共定义了1个对象

```
> x=2; setTimeout(console.log, 1000, x); x=3; x=4;
< 4   // x=4; 的返回值
  2   // console.log 打印的结果
```

```
> var x=[2,3,4];setTimeout(console.log,1000,x);x=
  [3,3,4];setTimeout(console.log,1000,x);x=
  [4,3,4];setTimeout(console.log,1000,x);x=
  [5,3,4];setTimeout(console.log,1000,x);
< 7 // 最后一次setTimeout的返回值
  ▶ (3) [2, 3, 4] // 第1次console.log的打印结果
  ▶ (3) [3, 3, 4] // 第2次console.log的打印结果
  ▶ (3) [4, 3, 4] // 第3次console.log的打印结果
  ▶ (3) [5, 3, 4] // 第4次console.log的打印结果
```

```
> var x=[2,3,4]; setTimeout(console.log, 1000, x); x[0]++; x[0]++;
< 3     // x[0]++; 的返回值
  ▶ (3) [4, 3, 4]  // console.log 打印的结果
```

JavaScript 动画

- 其它使用setTimeout的例子
  - for循环
    - 告知浏览器，1000ms后执行increase函数
    - 打印ar
      - increase尚未执行，ar不变
      - increase会在for循环执行完毕后再执行
  - increase函数
    - 1000ms后执行
    - 每次把ar[0]加1
      - 打印出的ar[0]递增

```
> var ar = [0,1,2,3];
  for(var i = 0; i < 5; i ++){
        setTimeout(increase, 1000, ar);
        console.log(ar);
  }
  function increase(ar){
        ar[0]++;
        console.log(ar);
  }
  ▶ (4) [0, 1, 2, 3]
  ▶ (4) [0, 1, 2, 3]
  ▶ (4) [0, 1, 2, 3]
  ▶ (4) [0, 1, 2, 3]
  ▶ (4) [0, 1, 2, 3]
<· undefined
  ▶ (4) [1, 1, 2, 3]
  ▶ (4) [2, 1, 2, 3]
  ▶ (4) [3, 1, 2, 3]
  ▶ (4) [4, 1, 2, 3]
  ▶ (4) [5, 1, 2, 3]
>
```

for循环中console.log的打印结果

increase函数中console.log的打印结果

第三周 提纲

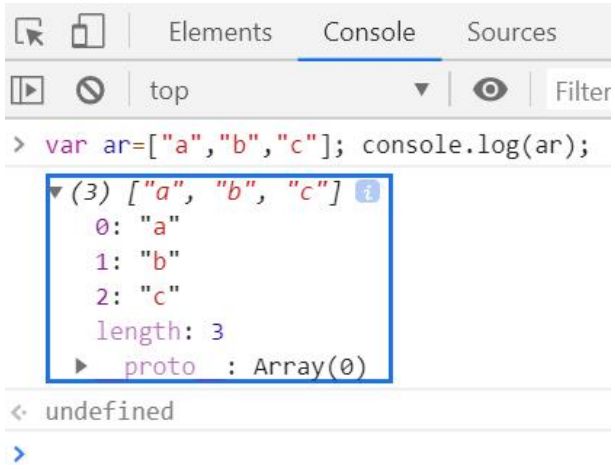
- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

Debug: console.log 函数

- console.log可以打印对象
  - 有时会打印对象的详细信息（属性、方法）
  - 很多JS内置函数的返回值是对象
  - 当不确定某函数的返回值时，可用console.log打印它
- 控制台可以执行用户直接输入的JS语句
- 在控制台输入下面的代码，然后按回车执行，观察控制台输出：

```
var ar = ["a", "b", "c"];
console.log(ar);
```

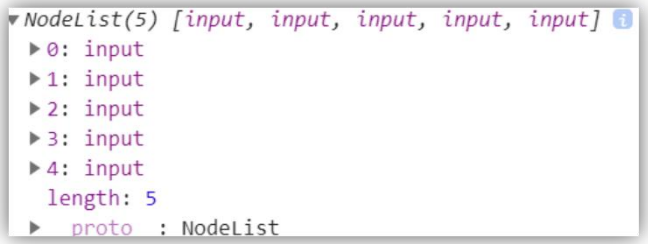
  - 图示使用Chrome浏览器



使用 console.log 的场景

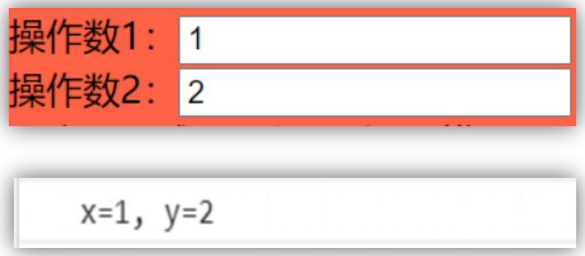
- 查看函数的返回值

```
var options =document.getElementsByName("op");
console.log(options);
```



- 打印中间变量的值

```
var operand1 = document.getElementById("operand1");
var x = operand1.value - 0;
var operand2 = document.getElementById("operand2");
var y = operand2.value - 0;
console.log("x=%d, y=%d", x, y);
```



可以使用 %d (round down),%f,%s 和%o(object)

第三周 提纲

- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

Go vs JS

- 要点
  - Go 是用来做“本地编程”的
    - 直接执行 I/O 操作
      - 操作硬盘中的文件
      - 读取终端的命令行输入
      - 可直接向终端输出信息
    - 所访问的资源均在本地
  - JS 是用来做web（网络）编程的
    - 通过浏览器执行 I/O 操作
      - 操作展示在浏览器里的网页
      - 向浏览器中的 console 里输出信息
    - 所访问的文件可在万维网（WWW）中

Go vs JS

	Go	JavaScript
变量	需要类型，一旦声明，类型不能变 如：var a int = 1	无类型，声明后可以把不同类型的值赋值给它，如：var a=1; a="JS";
数据类型	比JS多了数组、切片、字符	JS里没有切片，数组是通过“对象”实现的 JS里只有字符串类型，没有字符类型
函数	需要参数类型，返回值类型，可以有多个返回值	不需要参数类型，不需要写返回值类型，只能有一个返回值 函数是一种特殊“对象”，可组成函数数组
算术运算符	5/2=2　5.0/2=2.5 3.2 % 0.7 编译时报错	5/2=2.5 3.2 % 0.7 ≈ 0.4 (损失精度)
比较运算符	不同类型值的比较编译时会报错	如果用==(或!=)比较，不同类型的值先自动转换成相同类型，再比较 如果用===(或!==)比较，不同类型的值比较的结果是false(或true)
右移运算符 >>	A>>B A是无符号数：高位补0 A是有符号数：高位补符号位	A>>B：高位补0 A>>>B：高位补符号位

Go vs JS

	Go	JavaScript

控制流	条件不用加括号
	`if x<10 {`
	`……`
	`}`
		条件需要加括号
		`if (x<10){`
		`……`
		`}`

对象	没有讲	Array对象、Date对象、代表HTML元素的对象等
打印函数	`fmt.Println, fmt.Printf`	`console.log`
数组长度	`len(array)`	`array.length`
语言类型	编译型，需先编译成可执行文件，再运行可执行文件	解释型，浏览器内置的解释器一句句解释执行JS代码
语句	结尾不需要分号	结尾分号可有可无

Quicksort – Go vs JS

- Go
  - Go中创建切片、数组需要指定长度
  - Go可以传递切片，切片有长度，所以不需要传递待排数据的边界
  - fmt.Println 可直接输出到终端

```
func main() {
	rand.Seed
(time.Now().UnixNano())
	n :  = 1*1024*1024
	var da = make([]int,n)
	for i:  =0; i<n; i++ {
		da[i] = rand.Int()
	}
	quicksort (da[0:  n])
	fmt.Printf(" %d\n %d\n
%d\n",da[0],da[1],n-1,da[n-1])
}
```

- JS
  - JS里创建数组不需要指定长度
  - JS里生成随机数方式和Go不同
  - JS里没有切片，所以需要传递待排数据的边界
  - console.log 只能输出到控制台

Math.random返回[0,1)之间的随机数

```
<script>
  var ar = new Array();
  var n = 1*1024*1024;
  for(var i=0; i<n; i++){
    ar[i] = Math.floor(Math.random()*1000000);
  }
    quicksort(ar, 0, ar.length-1);
    console.log(" %d\n %d\n %d\n", ar[0],
ar[1], ar[ar.length-1]);
      ……
```

Quicksort – Go vs JS

- Go
  - if后面的条件不需要圆括号
  - array是一个切片，代表了原始数组的一段，可以直接用len(array)获取待排数据的长度
  - 划分函数partition只需要当前数组作参数

- JS
  - if后面的条件需要圆括号
  - 由于传递了数组的边界，所以需要通过边界来计算出待排数据长度
  - 划分函数partition还需要当前数组的边界作参数

```
func quicksort(array []int) {
  if len(array) < 2 {
    return
  }
  left_array, right_array := partition(array)
  quicksort(left_array)
  quicksort(right_array)
}
```

```
function quicksort(ar, left, right){
  if (right - left + 1 < 2) {
    return;
  }
  var pivotal_index=partition(ar, left, right);
  quicksort(ar, left, pivotal_index-1);
  quicksort(ar, pivotal_index+1, right);
}
```

Quicksort—Go vs JS

- Go

生成随机数的范围是[0, len(array)]

```
func partition(array []int) ([]int, []int)
{
 pivotal_index := rand.Intn(len(array))
 pivotal_value := array[pivotal_index]
 next := 0
 array[pivotal_index], array[len(array)-1]
=array[len(array)-1], array[pivotal_index]

 for i:  = 0; i < len(array); i++ {
   if (array[i] < pivotal_value) {
     array[next], array[i] =
     array[i], array[next]
     next++
   }
 }
 array[next], array[len(array)-1]=
 array[len(array)-1], array[next]
 return array[0: next], array[next+1:
len(array)]
}
```

可以使用同时给两个数赋值的语句来达到交换两个值的目的

返回两个切片

- JS

生成随机数的范围是[left, right]

```
function partition(ar, left, right){
      var len = right - left + 1;
      var pivotal_index =
Math.floor(Math.random()*len) + left;
      var pivotal_value =
ar[pivotal_index];
      var next=left;
      // pivotal to last
      ar[pivotal_index] = ar[right];
      ar[right] = pivotal_value;
      // smaller than pivotal :   left
      for(var i=left; i<=right; i++){
            if(ar[i] < pivotal_value) {
                  // swap
                  var temp = ar[next];
                  ar[next] = ar[i];
                  ar[i] = temp;
                  next++;
            }
      }
      // swap pivotal back
      ar[right] = ar[next];
      ar[next] = pivotal_value;
      return next;
}
```

交换两个值的时候，需要先把一个值存到另一个临时变量里

返回标杆下标

个人作品

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

补充材料

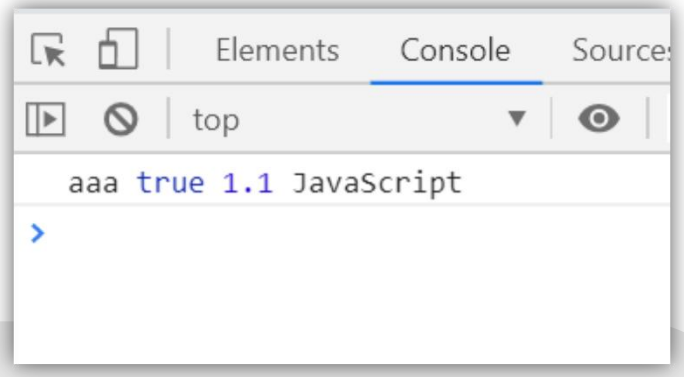
补充材料 提纲

- JavaScript 变量 & 函数
- JavaScript 操作符 & 表达式
- JavaScript 控制流
  - while 循环
  - if-else 语句
  - switch-case 语句

JavaScript 基础–基本数据类型

类型	说明	举例
布尔	表示真、假，只有两个值：true、false	true、false
数字	64bit长，可以同时表示浮点数和整数，相当于Go语言中float64	-1，0，1，0.1，-0.1
字符串	表示一个字符串	"abc"，""（空串）
null类型	null值属于一个只包含该值的特殊类型。 null表示此处不应该有值	null
undefined类型	undefined值属于一个只包含该值的特殊类型。 undefined表示缺少值： （1）变量声明后未赋值 （2）函数调用时没提供的参数 （3）函数没有返回值	undefined

JavaScript 编程知识–变量



- <script> 部分：

```
var x,y,z;      变量声明（不需要类型）
var s = "JavaScript";   声明并赋值
x = 0;          变量赋值
x = "aaa";      变量赋值（先后给同一个变量赋不同类型的值）
y= true;        变量赋值
z = 1.1;        变量赋值
console.log(x,y,z,s);  ⟶ 同时打印多个变量值，用逗号隔开
```

JavaScript 编程知识–函数

- 函数定义

相比Go，无需定义参数和返回值类型

JS:

```
function
add(x,y){
    return x + y;
}
```

Go:

```
func add(x,y int)
int{
    return x + y
}
```

函数体，只能返回至多一个值（而Go中可返回多个）

- 函数调用

```
var a=1,b=1;
var z = add(a,b); // z=2;
```

补充材料 提纲

- JavaScript 变量 & 函数
- JavaScript 操作符 & 表达式
- JavaScript 控制流
  - while 循环
  - if-else 语句
  - switch-case 语句

JavaScript 运算符 & 表达式

- 与Go相同的算术运算符
  - +,-,*,++,--
- 与Go不同的算术运算符
  - /（除法）
    - JS中：3/2=1.5，Go中：3/2=1（向下取整）
    - 如果JS中要实现向下取整，需要使用Math.floor(3/2)
  - %（取余）
    - JS中 % 的两操作数可以有小数，而Go不允许这样做。
    - JS中：3.2 % 0.7 ≈ 0.4(有精度损失)，Go中：不允许 3.2%0.7

JavaScript 运算符 & 表达式

- JS里运算符不只有算术运算符，Go里有的几乎都能在JS里找到对应的：
- 赋值运算符
  - =，+=，-=，*=，/=，%=
  - x op= y相当于x=x op y，如x+=y相当于x=x+y
    - op可以是+、-、*、/、%

```
> console.log(false==0)
  true
< undefined
> console.log(false===0)
  false
< undefined
```

- 比较运算符
  - 比较运算的结果是true或false
    - 与Go相同：>,>=,<,<=
    - 与Go不同：==,!=,===,!==
      - 使用==或!=比较时，若两被比较数类型不同，先自动转换成相同类型再比较
      - 使用===(!==)比较时，若两被比较数类型不同，则比较结果为false(true)

JavaScript 运算符 & 表达式

- 逻辑运算符
  - 与Go中相同：&&（与），||（或），!（非）
  - 经常在**条件表达式**里使用
  - Eg: if(x<10 && x>0) { … }

- 位运算符
  - 与Go相同：&（按位与），|（按位或），^（按位异或），~（按位取反），<<（左移）
  - 与Go不同：
    - >>（右移，左边补符号位，负数补1，非负数补0）
    - >>>（右移，左边补0）
  - 当进行位运算时，参与运算的整数被当作32位整数

JavaScript 运算符 & 表达式

- 条件运算符（一种三元运算符）
  - 条件 ？条件为真时表达式的值 ： 条件为假时表达式的值
  - 比如：`var y = x>0?x:-x; //y=x的绝对值`

- 字符串连接运算符
  - +（加号，和加法用同一个）
  - 当两个操作数至少有一个是字符串时，将不是字符串的操作数转化成字符串，然后连接：

```
var x = "Java" + "Script"; //x="JavaScript"
var x = "0" + 1;  //x= "01" （1先被转换成字符串）
```

补充材料 提纲

- JavaScript 变量 & 函数
- JavaScript 操作符 & 表达式
- JavaScript 控制流
  - while 循环
  - if-else 语句
  - switch-case 语句

JavaScript 控制流—while 循环

- while 循环：

```
while(表达式){
    // 一些代码
}
```

- 每次循环，若表达式为真，则执行循环体；否则跳过循环体，停止循环。

JavaScript 控制流–if-else 语句

- if-else 语句：
  - 同Go类似，但注意表达式需要加圆括号。

```
if(表达式1){
    //一系列语句
}else if (表达式2){
    //一系列语句
}else{
    //一系列语句
}
```

可以有零个或
多个else分支

JavaScript 控制流– switch-case语句

- switch-case 语句:
  - 可代替 if-else 语句

```
switch(表达式){
    case 值1: //一系列代码，当表达式的值等于值1时执行
    break;    //不要忘记break;否则，已匹配case语句后面所有语句都会无条件执行，
              //直到碰到后面的break或者到达整个switch-case语句末尾
    case 值2: //一系列代码，当表达式的值不等于值1且等于值2时执行
    break;
    default:  //一系列代码，当表达式的值既不等于值1也不等于值2时执行
                  //default可以省略
}
```