JavaScript对象

对象

面向对象和面向过程是两种不同的程序设计思想。面向过程的程序设计中，数据和对数据的操作分别放在变量和函数中，不能构成一个整体。而面向对象的程序设计中，将数据和数据的操作封装成一个对象，所有事物都可以表示成对象，例如一门课程，一个用户，一次与远程服务器的连接等等都可以看作对象。

JavaScript的对象本质上是无序属性的集合，属性名和属性值组成键值对，属性值可以是数据和函数。为了区分，常常把数据属性称为属性，函数属性称为方法。属性和方法统称为成员。对象的静态特征用属性表示，动态特征用方法表示。例如一个学生可以表示成一个对象，学生具有学号、姓名、入学时间、身份（本科生/研究生）、已修学分等等属性，并且具有选课、上课、考试等方法。

JavaScript还提供多个标准内置对象，例如Object、Array、Date、Math、String等。此外，根据需要，JavaScript还可以自定义对象。

对象

1. 什么是对象
2. 对象使用
3. 操作对象
4. 遍历对象
5. 内置对象

什么是对象

- 对象（object）：JavaScript里的一种数据类型
- 包含相关数据和方法的集合（通常由一些变量和函数组成，我们称之为对象里面的属性和方法）
- 用来描述某个事物，例如描述一个人
  - 人有姓名、年龄、性别等信息、还有吃饭睡觉听音乐、看电影、运动等等功能
  - 如果用多个变量保存则比较散，用对象来描述会比较统一
- 比如描述 “老师” 信息：
  - 静态特征（姓名，年龄，身高，性别，爱好）=> 可以使用数字，字符串，数组，布尔类型等表示
  - 动态行为（点名，唱歌，运 动）=> 使用函数表示

对象的使用

掌握对象语法，用它保存多个数据

```
> var o3={}
< undefined

> o3
< ▼{} i
    ▶[[Prototype]]: Object
```

对象的使用

掌握对象语法，用它保存多个数据

```
> var person = {
    name : ['Bob', 'Smith'],
    age : 32,
    gender : 'male',
    interests : ['music', 'skiing'],
    bio : function() {
      alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years
  old. He likes ' + this.interests[0] + ' and ' + this.interests[1] + '.');
    },
    greeting: function() {
      alert('Hi! I\'m ' + this.name[0] + '.');
    }
  };
```

- 属性都是成对出现的，包括属性名和值，它们之间使用英文：分隔
- 多个属性之间使用英文，分隔
- 属性名可以使用 ”” 或 ’’，一般情况下省略，除非名称遇到特殊符号如空格、中横线等

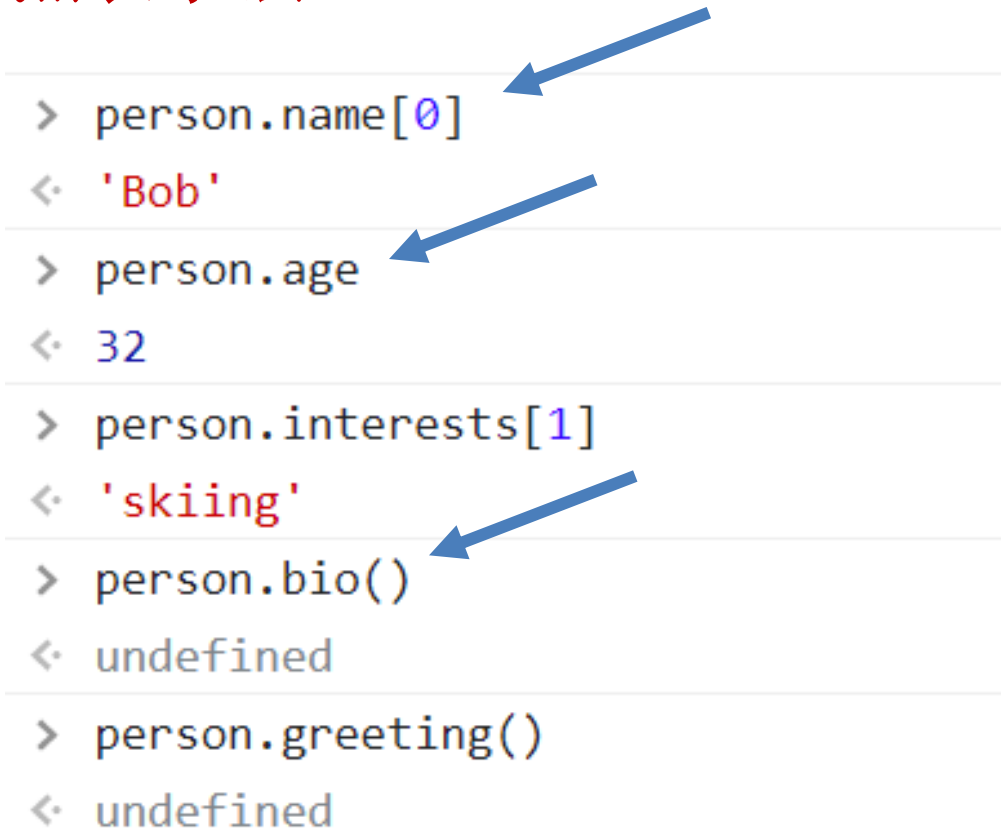
对象的使用

掌握对象语法，用它保存多个数据

```
> person.name[0]
'Bob'
> person.age
32
> person.interests[1]
'skiing'
> person.bio()
undefined
> person.greeting()
undefined
```

对象的使用

点表示法


> person.name[0]
'Bob'
> person.age
32
> person.interests[1]
'skiing'
> person.bio()
undefined
> person.greeting()
undefined

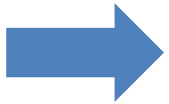
对象的使用

子命名空间

可以用一个对象来做另一个对象成员的值。例如将 name 成员改写一下：

```
name : ['Bob', 'Smith'],
```

```
name : {
  first : 'Bob',
  last : 'Smith'
},
```



```
person.name.first
person.name.last
```

实际上创建了一个子命名空间　　链式的再使用一次点表示法

对象的使用

中括号表示法

另外一种访问属性的方式是使用中括号表示法 (bracket notation)。

```
person.age
person.name.first
```

```
person['age']
person['name']['first']
```

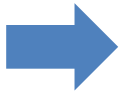
对象有时被称之为关联数组 (associative array) ——对象做了字符串到值的映射，而数组做的是数字到值的映射。

对象的使用

设置对象成员

除了可以访问对象的成员，也可以设置对象成员的值，声明要设置的成员：

```
person.age = 45
person['name']['last'] = 'Cratchit'
```



```
> person.age
< 45
> person['name']['last']
< 'Cratchit'
```

设置成员并不意味着只能更新已经存在的属性的值，完全可以创建新的成员：

```
person['eyes'] = 'hazel'
person.farewell = function() { alert("Bye everybody!") }
```

```
> person['eyes']
< 'hazel'
> person.farewell()
```

developer.mozilla.org 显示

Bye everybody!

确定

对象的使用

设置对象成员

括号表示法一个有用的地方是它不仅可以动态的去设置对象成员的值，还可以动态的去设置成员的名字：

person[myDataName] = myDataValue

例如：

```
> var myDataName = 'height'
<- undefined
> var myDataValue = '1.75m'
<- undefined
> person[myDataName] = myDataValue
<- '1.75m'
```

```
> person.height
<- '1.75m'
```

使用点表示法无法做到上述，点表示法只能接受字面量的成员的名字，不接受变量作为名字。

对象的使用

“this”含义

关键字“this”指向了当前代码运行时的对象。

保证了当代码的上下文 (context) 改变时变量的值的正确性（比如：不同的 person 对象拥有不同的 name 这个属性，很明显 greeting 这个方法需要使用的是对象自己的 name）。

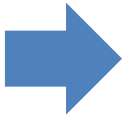
```
> var person = {
    name : ['Bob', 'Smith'],
    age : 32,
    gender : 'male',
    interests : ['music', 'skiing'],
    bio : function() {
      alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years
old. He likes ' + this.interests[0] + ' and ' + this.interests[1] + '.');
    },
    greeting: function() {
      alert('Hi! I\'m ' + this.name[0] + '.');
    }
};
```

对象的使用

“this”含义

```
var person1 = {
  name : 'Chris',
  greeting: function() {
    alert('Hi! I\'m ' + this.name + '.');
  }
}

var person2 = {
  name : 'Brian',
  greeting: function() {
    alert('Hi! I\'m ' + this.name + '.');
  }
}
```



person1.greeting() 会输出："Hi! I'm Chris."；

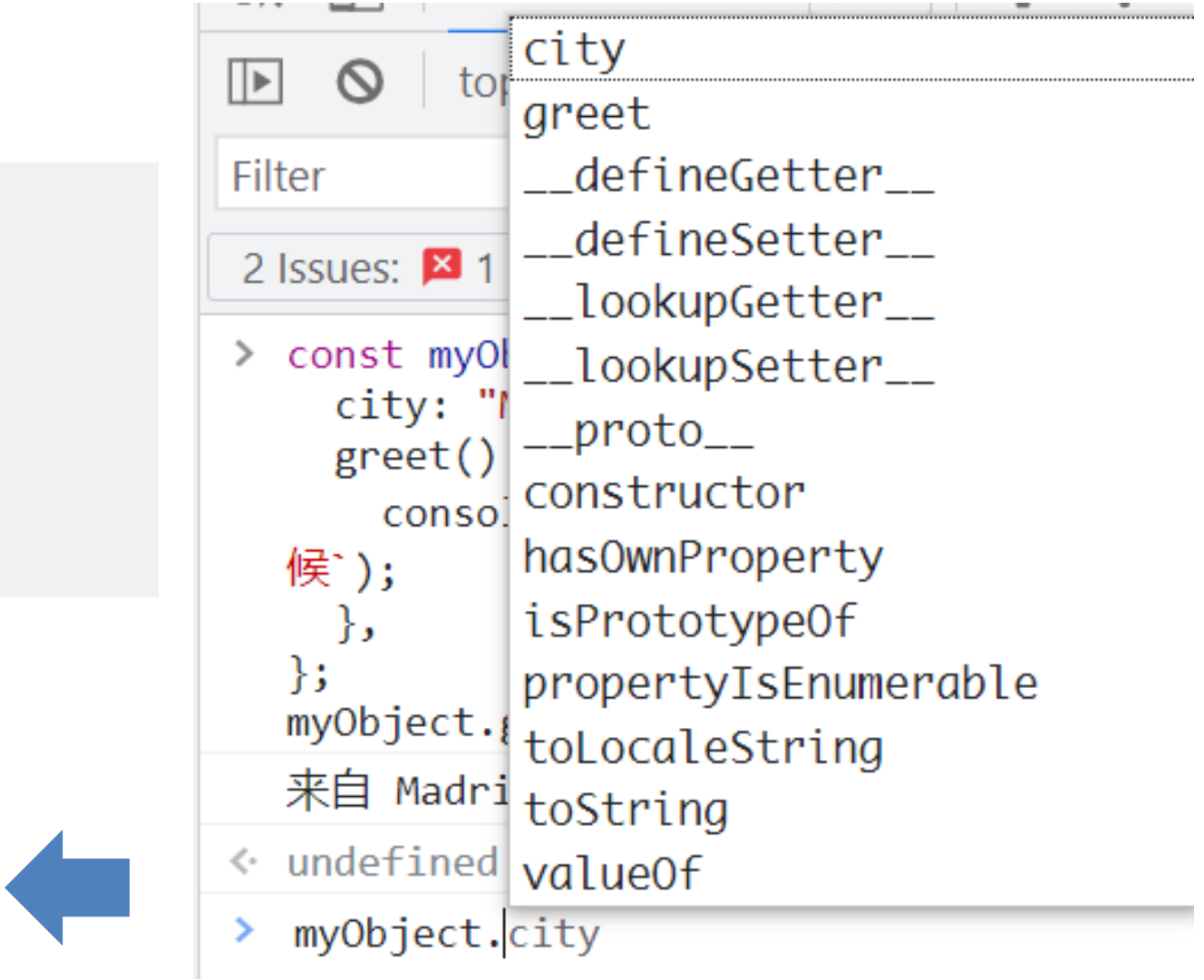
person2.greeting() 会输出："Hi! I'm Brain."

对象原型

原型链

```
const myObject = {
  city: "Madrid",
  greet() {
    console.log(`来自 ${this.city} 的问候`);
  },
};
myObject.greet(); // 来自 Madrid 的问候
```

```
> myObject.toString()
'[object Object]'
```



对象原型

原型链

这些额外的属性是什么，它们是从哪里来的？

- JavaScript 中所有的对象都有一个内置属性，称为它的 prototype（原型）。它本身是一个对象，故原型对象也会有它自己的原型，逐渐构成了原型链。原型链终止于拥有 null 作为其原型的对象上。

- 当试图访问一个对象的属性时：如果在对象本身中找不到该属性，就会在原型中搜索该属性。如果仍然找不到该属性，那么就搜索原型的原型，以此类推，直到找到该属性，或者到达链的末端，在这种情况下，返回 undefined。

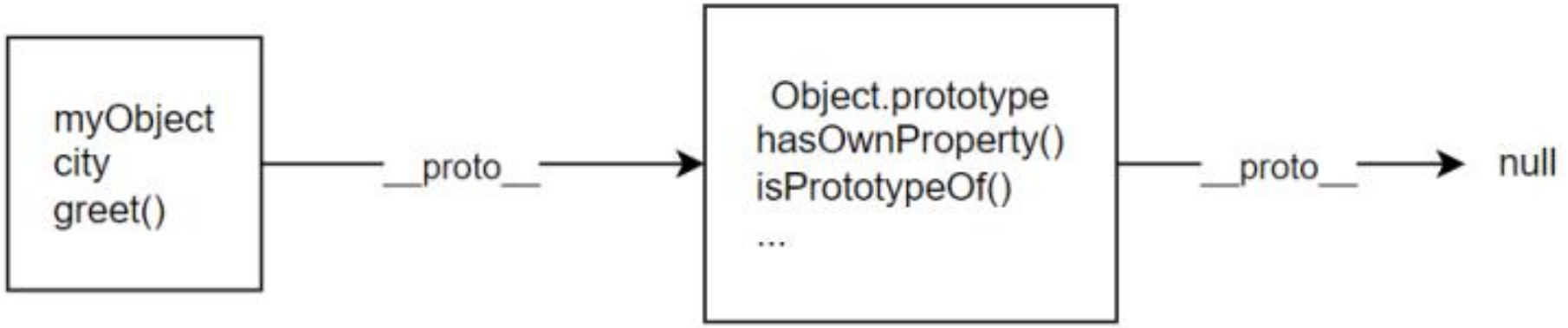
对象原型

原型链

在调用 myObject.toString() 时，浏览器做了这些事情：

1. 在 myObject 中寻找 toString 属性
2. myObject 中找不到 toString 属性，故在 myObject 的原型对象中寻找 toString
3. 其原型对象拥有这个属性，然后调用它。

使用 Object.getPrototypeOf() 函数查看myObject 的原型：



有个对象叫 Object.prototype，它是最基础的原型，所有对象默认都拥有它。Object.prototype 的原型是 null，所以它位于原型链的终点。

对象原型

原型链

```
> Object.getPrototypeOf(myObject)
< {constructor: ƒ, __defineGetter__: ƒ, __defineSetter__: ƒ, hasOwnProperty: ƒ, __lookupGetter__: ƒ, …}
    constructor: ƒ Object()
    hasOwnProperty: ƒ hasOwnProperty()
    isPrototypeOf: ƒ isPrototypeOf()
    propertyIsEnumerable: ƒ propertyIsEnumerable()
    toLocaleString: ƒ toLocaleString()
    toString: ƒ toString()
    valueOf: ƒ valueOf()
    __defineGetter__: ƒ __defineGetter__()
    __defineSetter__: ƒ __defineSetter__()
    __lookupGetter__: ƒ __lookupGetter__()
    __lookupSetter__: ƒ __lookupSetter__()
    __proto__: null
    get __proto__: ƒ __proto__()
    set __proto__: ƒ __proto__()
>
```

对象原型

原型链

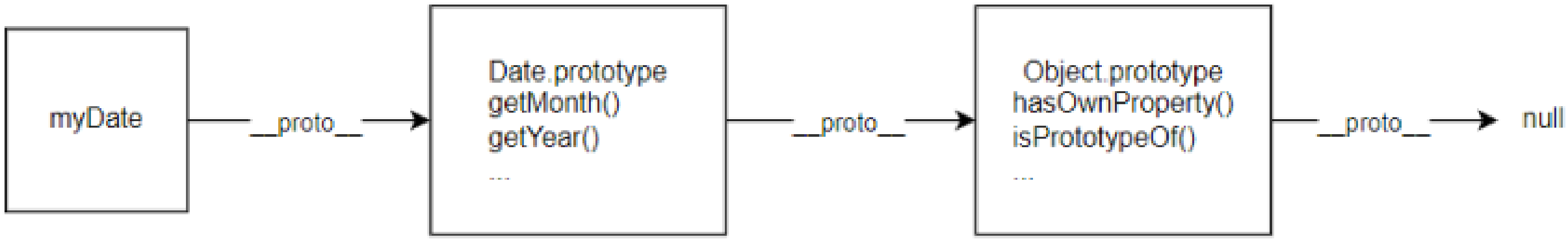
```
const myDate = new Date();
let object = myDate;

do {
  object = Object.getPrototypeOf(object);
  console.log(object);
} while (object);

// Date.prototype
// Object { }
// null
```

一个对象的原型并不总是 Object.prototype。

左边例子创建了 Date 对象，遍历了它的原型链，记录并输出了原型。由此可见myDate 的原型是 Date.prototype 对象， Date.prototype的原型是 Object.prototype。



对象原型

属性覆盖

```
const myDate = new Date(1995, 11, 17);

console.log(myDate.getYear()); // 95

myDate.getYear = function () {
  console.log("别的东西! ");
};

myDate.getYear(); // '别的东西! '
```

调用 getYear() 时，浏览器首先在 myDate 中寻找具有该名称的属性，如果 myDate 没有定义该属性，才检查原型。因此，当我们给 myDate 添加 getYear() 时，就会调用 myDate 中的版本。

对象原型

设置原型—Object.create()

在 JavaScript 中，有多种设置对象原型的方法，介绍两种Object.create() 和构造函数。

```
const personPrototype = {
  greet() {
    console.log("hello!");
  },
};

const carl = Object.create(personPrototype);
carl.greet(); // hello!
```

对象原型

设置原型—构造函数

- 在 JavaScript 中，所有的函数都有一个名为 prototype 的属性。当调用一个函数作为构造函数时，这个属性被设置为新构造对象的原型（按照惯例，在名为 __proto__ 的属性中）。

```
const personPrototype = {
  greet() {
    console.log(`你好，我的名字是 ${this.name}！`);
  },
};

function Person(name) {
  this.name = name;
}

Object.assign(Person.prototype, personPrototype);
// 或
// Person.prototype.greet = personPrototype.greet;
```

1. 创建了一个 personPrototype 对象，它具有 greet() 方法
2. 创建了一个 Person() 构造函数，它初始化了要创建人物对象的名字

然后我们使用 Object.assign 将 personPrototype 中定义的方法绑定到 Person 函数的 prototype 属性上。
在这段代码之后，使用 Person() 创建的对象将获得 Person.prototype 作为其原型，其中自动包含 greet 方法。

```
const reuben = new Person("Reuben");
reuben.greet(); // 你好，我的名字是 Reuben！
```

对象原型

自有属性

上面的 Person 构造函数创建的对象有两个属性：

name 属性，在构造函数中设置，在 Person 对象中可以直接看到

greet() 方法，在原型中设置。

直接在对象中定义的属性，如这里的name，被称为自有属性。自有属性可以通过Object.hasOwn()方法检测。

```
const irma = new Person("Irma");

console.log(Object.hasOwn(irma, "name")); // true

console.log(Object.hasOwn(irma, "greet")); // false
```

JSON--一种按照 JavaScript 对象语法的数据格式

JavaScript 对象表示法（JSON）是用于将结构化数据表示为 JavaScript 对象的标准格式，通常用于在网站上表示和传输数据（例如从服务器向客户端发送一些数据，因此可以将其显示在网页上）

JSON基于 JavaScript 语法，但它独立于 JavaScript，这也是为什么许多程序环境能够读取（解读）和生成 JSON。

JSON结构

```
{
  "squadName" : "Super hero squad",
  "homeTown" : "Metro City",
  "formed" : 2016,
  "secretBase" : "Super tower",
  "active" : true,
  "members" : [
    {
      "name" : "Molecule Man",
      "age" : 29,
      "secretIdentity" : "Dan Jukes",
      "powers" : [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name" : "Madame Uppercut",
      "age" : 39,
      "secretIdentity" : "Jane Wilson",
      "powers" : [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name" : "Eternal Flame",
      "age" : 1000000,
      "secretIdentity" : "Unknown",
      "powers" : [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

如果想访问JSON对象中的对象，通过链式访问（属性名和数组索引）

比如访问 superHeroes 对象中的 members 数组对象的第二个元素的 powers 数组对象的第三个元素：

```
superHeroes["members"][1]["powers"][2]
```

➡ 'Superhuman reflexes'

JSON对象数组

```
[
  {
    "name" : "Molecule Man",
    "age" : 29,
    "secretIdentity" : "Dan Jukes",
    "powers" : [
      "Radiation resistance",
      "Turning tiny",
      "Radiation blast"
    ]
  },
  {
    "name" : "Madame Uppercut",
    "age" : 39,
    "secretIdentity" : "Jane Wilson",
    "powers" : [
      "Million tonne punch",
      "Damage resistance",
      "Superhuman reflexes"
    ]
  }
]
```

通过数组索引访问数组元素

注意：

1. JSON 是一种纯数据格式，它只包含属性，没有方法。
2. JSON 要求在字符串和属性名称周围使用双引号。单引号无效。
3. 甚至一个错位的逗号或分号就可以导致 JSON 文件出错。
4. 与 JavaScript 代码中对象属性可以不加引号不同，JSON 中只有带引号的字符串可以用作属性。

操作对象

对象本质是无序的数据集合，操作数据无非就是 增 删 改 查 语法：

查询对象

对象.属性

对象[‘属性’]

对象.方法()

查

改

重新赋值

对象.属性 = 值

对象.方法 = function() {}

对象添加新的数据

对象名.新属性名 = 新值

增

删

删除对象中属性

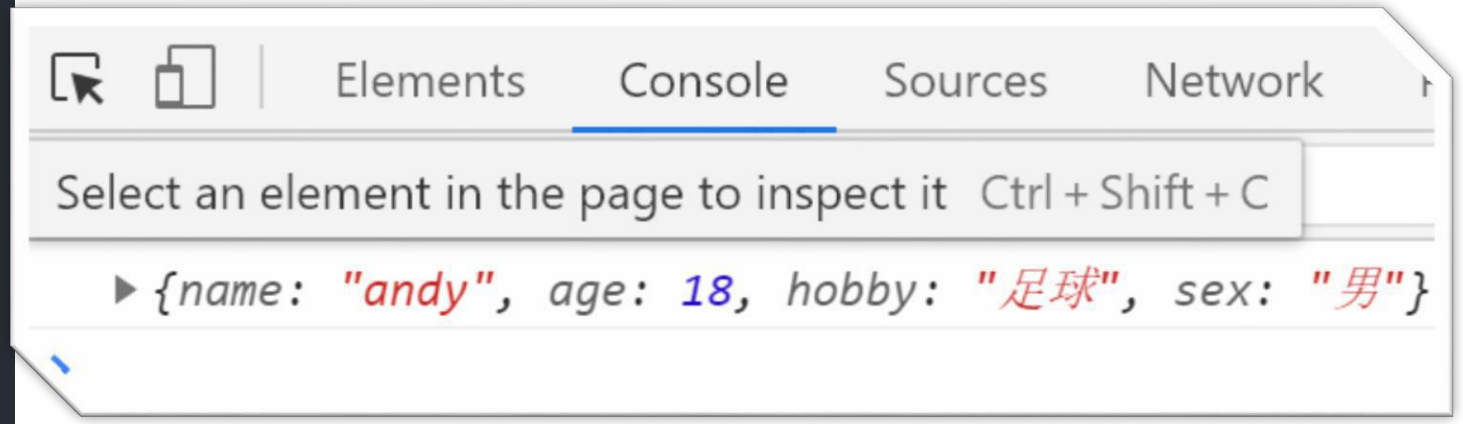
delete 对象名.属性名

操作对象

增加属性

也可以动态为对象添加属性，动态添加与直接定义是一样的，只是语法上更灵活。

```
let person = {
    name: 'andy',
    age: 18,
}
person.hobby = '足球'
person['sex'] = '男'
console.log(person)
```



操作对象

新增对象中的方法

也可以动态为对象添加方法，动态添加与直接定义是一样的，只是语法上更灵活

```
person.move = function() {
    document.write('移动一点点')
}
```

注：无论是属性或是方法，同一个对象中出现名称一样的，后面的会覆盖前面的。

遍历对象

遍历对象元素

对象没有像数组一样的length属性,所以无法确定长度

对象里面是无序的键值对，没有规律，不像数组里面有规律的下标

```
let obj = {
    uname: 'andy',
    age: 18,
    sex: '男'
}
for (let k in obj) {
    console.log(k)  // 打印属性名
    console.log(obj[k])  // 打印属性值
}
```

- 一般不用这种方式遍历数组、主要是用来遍历对象；
- 一定记住：**k** 是获得对象的**属性名**，**对象名[k]** 是获得 **属性值；**

遍历对象

案例 遍历数组对象

需求：请把下面数据中的对象打印出来：

```
// 定义一个存储了若干学生信息的数组
let students = [
  {name: '小明', age: 18, gender: '男', hometown: '河北省'},
  {name: '小红', age: 19, gender: '女', hometown: '河南省'},
  {name: '小刚', age: 17, gender: '男', hometown: '山西省'},
  {name: '小丽', age: 18, gender: '女', hometown: '山东省'}
]
```

遍历对象

案例 遍历数组对象

需求：根据以上数据渲染生成表格

1. 打印表格 头部和尾部
2. 中间的行遍历数组，然后填充对象数据

学生列表

序号	姓名	年龄	性别	家乡
1	小明	18	男	河北省
2	小红	19	女	河南省
3	小刚	17	男	山西省
4	小丽	18	女	山东省

内置对象

掌握内置对象，调用JavaScript准备好的功能

1、内置对象是什么

JavaScript内部提供的对象，包含各种属性和方法给开发者调用

2、内置对象Math、Date、String

内置对象

2 内置对象Math

- Math对象是JavaScript提供的一个“数学高手”对象
- 提供了一系列做数学运算的方法
- 方法有：
  - random：生成0-1之间的随机数（包含0不包括1）
  - ceil：向上取整
  - floor：向下取整
  - max：找最大数
  - min：找最小数
  - pow：幂运算
  - abs：绝对值
  - Math对象在线文档：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Math

内置对象

生成任意范围随机数

- 如何生成0-10的随机数呢？

```
Math.floor(Math.random() * (10 + 1))
```

- 如何生成5-10的随机数？

```
Math.floor(Math.random() * (5 + 1)) + 5
```

- 如何生成N-M之间的随机数

```
Math.floor(Math.random() * (M - N + 1)) + N
```

内置对象

案例 应用随机函数，随机生成0~100的整数，显示在表格中，每次刷新页面表格中的数据都不同。

18	73	86
64	17	58

内置对象

案例

猜数字游戏

需求：程序随机生成 1 到 100 之间的随机数，用户在输入框输入猜测的数字，系统会给出大还是小的提示，只能猜测10次，10次还没猜对，游戏就结束。

内置对象

4、Date对象

Date对象用于处理日期和时间，是基于1970年1月1日（世界标准时间）起的毫秒数表示时间的。

1、Date对象的构造函数

（1）构造函数Date()

可以通过下边方法获取当前系统时间，例如：

```
let now = new Date()
console.log(now)
```

```
Sat Nov 19 2022 17:46:49 GMT+0800 (中国标准时间)
```

（2）构造函数Date(毫秒)

```
let d = new Date(188980399999)
console.log(d)
```

```
Sun Dec 28 1975 14:33:19 GMT+0800 (中国标准时间)
```

（3）构造函数Date(字符串)

```
let d = new Date('2022-11-19 14:00:23')
console.log(d)
```

```
Sat Nov 19 2022 14:00:23 GMT+0800 (中国标准时间)
```

内置对象

Date对象常用成员

getDate()：获取日期（1-31）

getFullYear()：获取年份（yyyy）

getMonth()：获取月（0-11）

getDay()：获取星期几（0-6）

getHours()：获取小时（0-23）

getMinutes()：获取分钟（0-59）

getSeconds()：获取秒（0-59）

valueOf()：获取原始毫秒值

setDate()：设置日期

setFullYear()：设置年份（四位数字）

setMonth()：设置月（0-11）

----

setHours()：设置小时（0-23）

setMinutes()：设置分（0-59）

setSeconds()：设置秒（0-59）

toString()：Date对象转换为字符串

例如，输出当年系统时间的英文月份：

```
let d = new Date()
const months = ['January', 'February', 'March', 'April', 'May', 'June',
'July', 'August', 'September', 'October', 'November', 'December']
let m = d.getMonth()
document.write(months[m])
```

内置对象

5、String对象

String对象用于保存和处理字符串，可以通过new String创建字符串对象。

```
let str = new String('内置对象String')
```

String对象常用成员：

length： 字符串长度； **//str.length-> 10**

indexOf()：检索元素首次出现的位置，检索不到返回-1； **//str.indexOf('对象')-> 2**

lastIndexOf():反向检索元素最后一次出现的位置； **//str.lastIndexOf('对象')-> 2**

charAt()：返回在指定位置的字符； **//str.charAt(2)-> "对"**

substr()：提取子串，需给定起始位置、长度； **//str.substr(2,2)-> "对象"**

substring()：提取子串，需给定起始位置、终止位置，自动比较位置值的大小，截取子串。 **//str.substring(0,5)->"内置对象S"**

slice()：提取子串，需给定起始位置、终止位置，接受负数索引表示。 **//str.slice(-4,-2)->"ri"**

replace()：替换子串。 **//str.replace("内置","JS")->"JS对象String"**

内置对象

5、String对象

String对象常用成员：

concat()： 连接字符串； **//str.concat(“123”)->”内置对象String123”**

split()：将字符串分割为字符串数组； **//let str=“web, JS, CSS”; str.split(‘,’) ->[‘web’, ‘JS’, ‘CSS’]**

toLowerCase():将字符串转换为小写；

toUpperCase():将字符串转换为大写；

toString()：返回字符串，Array、Number等都可以调用这个，将其转换为字符串；

valueOf(): 返回某个字符串对象的原始值。

内置对象

案例

应用String对象的方法，对字符串中的字符进行计数，并输出计数最大的字符及其出现次数。例如，对于字符串‘Hello JS.’，输出结果为2和1（字母l）。

```
let s = 'Hello js.'
s = s.toLowerCase()
//定义charCnt对象，包含字母和计数键值对
let charCnt = {}
for (let i = 0; i < s.length; i++) {
    let c = s.charAt(i)
    if (charCnt[c]) {
        charCnt[c]++
    } else {
        charCnt[c] = 1
    }
}
//在charCnt中查找计数最大的字符
let max = 0
let charOfMax
for (let key in charCnt) {
    if (max < charCnt[key]) {
        max = charCnt[key]
        charOfMax = key
    }
}
console.log(max)
console.log(charOfMax)
```

总结

对象包含属性和方法，分别表示事物的静态特征和动态特征。JavaScript支持自定义对象和使用系统预先定义好的标准内置对象。

创建对象可以通过字面量、new Object()、构造函数或者原型的方式。简单对象应用可用字面量和new Object()创建，但是创建多个对象时使用构造函数代码更加简洁，通过构造函数或原型创建对象将进一步降低内存消费。对象的属性和方法可以通过“.”进行引用并支持动态添加或删除。

除自定义对象外，JavaScript还提供Math对象用于数学运算，Date对象用于处理日期和时间，String对象用于处理字符串。