第一章JavaScript数据类型

1 基本使用

使用方法

- 直接嵌入标签当中。type属性默认是javascript

```
<head>
<script type="text/javacript">

</script>
</head>
```

- 单独放入js文件当中，通过标签引入

```
<head>
<script src="/static/js/abc.js">
</head>
```

2 基本语法

> 单独语句尽量加分号，自动加分号会导致理解问题。

赋值语句

```
var x = 1;
```

注释语句

```
//单行注释
/* 多行注释
多行注释*/
```

3 数据类型

Number

> 不区分证书和浮点数

```
123//整数
-0.45//负数、浮点数
1.23e3//科学计数法
NaN//不是数字
Infinity//无限大
```

bool值

```
true;
false;
```

null与undefined

```
null; \\表示空值，不等于0，不等于''空字符串。
undefined;\\未定义，没什么用
```

字符串

```
'hello';
"world";
"\"\u";//反斜杠转义字符，Unicode字符
`h
ll
o`;//反引号多行字符串
```

数组array（列表list，集合set）

```
[1,2,3.14,'hello',null,true];
new Array(1,2,3);
var arr =[1,2,3.14,'hello',null,true];
arr[0];
```

对象object（Dictionary字典，map映射）

```
var person ={
      name:'bob',
      age:20,
      tags:['js','web'],
      zipcode:null
};
```

4 数据形式

变量定义var

```
var a;//不强调类型
var b = 1;
```

常量定义const

const来定义常量

```
const PI = 3.14;
PI = 3; // 某些浏览器不报错，但是无效果！
PI; // 3.14
```

4 字符串

模板字符串

```
var name = 'xiaoming';
var message= "hello" + "world"+name;//两种字符串拼接的办法，通过加号连接。
var message = 'hello world ${name}';//两种字符串拼接的方法，模板字符串
```

字符串访问

```
var s = 'Hello,world!';
s.length//长度属性，字符串看成对象。
s[0]//访问字符，字符串看成列表、集合、数组。
s[0]='t';//下标只能访问，不能修改字符串，字符串是不可变的。
```

字符串操作

```
var s='Hello';
s.toUpperCase();
s.toLowerCase();
s.indexOf('lo');//字符串首次出现的位置。
s.substring(0,5);//字字符串
```

5 数组

数组访问与修改

```
var arr = [1,2,3];
arr.length;//=3
arr.length=6;
arr[1]=99;//数组内容与长度均可变。长度变为6
```

> 如果通过索引赋值时，索引超过了范围，同样会引起Array大小的变化：

```
var arr = [1, 2, 3];
arr[5] = 'x';
arr; // arr变为[1, 2, 3, undefined, undefined, 'x']
```

数组的操作

```
var arr = [10,20,'30','xyz'];
arr.indexOf(10);
arr.slice(0,3);//切片操作
arr.push('a','b');//向末尾加入多个元素
arr.pop();//去掉最后一个元素
arr.unshift('a','b');//向开头加入多个元素
arr.shift();//去掉第一个元素并返回。
arr.sort();//默认方法排序。
arr.reverse();//array 反转
arr.splice(2,3,'google',88);//从指定位置开始删除若干元素，然后，添加若干元素。
var added = arr.concat([1,2,3]);//连接另一个数组，并返回新的数组
arr.join('-');//用指定字符连接数组元素
```

多维数组

```
var arr= [[1,2,3],[400,500,600],'0'];
arr[1][1];//=500
```

6 对象

对象访问

```
var xiaoming = {
      name:'xiaoming',
      birth:1990,
      'weight':333
};

xiaoming.name;//'xiaoming'通过点运算符访问。
xiaohong['middle-school'];//访问特殊字符的键。只能用下标的方式。
```

对象修改

```
var xiaoming = {
      name: '小明'
};
xiaoming.age; // undefined
xiaoming.age = 18; // 新增一个age属性
xiaoming.age; // 18
delete xiaoming.age; // 删除age属性
xiaoming.age; // undefined
delete xiaoming['name']; // 删除name属性
xiaoming.name; // undefined
delete xiaoming.school; // 删除一个不存在的school属性也不会报错

'name' in xiaoming;//检查属性是否存在。
```

7 运算符

> NaN与NaN也不相等。使用isNaN()判断是否为数值类型。

```
+ - * / %;//基本运算符
> < >= <= ==（强制类型转换） ===（包括类型）;//比较运算符
! && || ;//逻辑运算符
& | ~ ^ << >> >>>//位运算符
typeof//变量类型运算符
instanceof//对象类型运算符
```

8 Map映射

创建map

```
var m = new Map([['Michael', 95], ['Bob', 75], ['Tracy', 85]]);
m.get('Michael');
```

map使用

```
var m = new Map(); // 空Map
m.set('Adam', 67); // 添加新的key-value
m.set('Bob', 59);
m.has('Adam'); // 是否存在key 'Adam': true
m.get('Adam'); // 67
m.delete('Adam'); // 删除key 'Adam'
m.get('Adam'); // undefined
```

9 Set集合

创建Set

单值，值不能重复

```
var s1 = new Set(); // 空Set
var s2 = new Set([1, 2, 3]); // 含1, 2, 3
```

使用Set

```
s.add(4);
s.delete(3);
```

10 iterable循环类型

array、map、Set、object都可以通过for of进行循环。但array本身也是对象，也是可以添加新的属性的。

for-of

```
var a = ['A', 'B', 'C'];
var s = new Set(['A', 'B', 'C']);
var m = new Map([[1, 'x'], [2, 'y'], [3, 'z']]);
for (var x of a) { // 遍历Array
      console.log(x);
}
for (var x of s) { // 遍历Set
      console.log(x);
}
for (var x of m) { // 遍历Map
      console.log(x[0] + '=' + x[1]);
}
```

foreach

接收一个函数，每次迭代就自动回调该函数

```
var s = new Set(['A', 'B', 'C']);
s.forEach(function (element, sameElement, set) {
      console.log(element);
});
```

> 区别复合数据类型
>
> - Array 数组，有顺序的单值，单值可重复。
> - Set 集合，无顺序的单值，值不可以重复。
> - object对象，无顺序的键值对，键必须是字符串。
> - Map映射，无顺序的键值对，键可以是任意类型。
> - iterable，加在其他类型上的特殊方法。

第二章 JavaScript程序结构

1 条件判断

if-else

```
var age = 20;
if (age >= 18) { // 如果age >= 18为true，则执行if语句块
      alert('adult');
} else { // 否则执行else语句块
      alert('teenager');
}
//else是可选的
//如果语句块只有一句话，可以省略大括号。
```

if-else可以嵌套

```
var age = 3;
if (age >= 18) {
      alert('adult');
} else {
      if (age >= 6) {
            alert('teenager');
      } else {
            alert('kid');
      }
}

if (age >= 6) {
      console.log('teenager');
} else if (age >= 18) {
      console.log('adult');
} else {
      console.log('kid');
}
```

2 循环

for循环

```
var x = 0;
var i;
for (i=1; i<=10000; i++) {
      x = x + i;
}
```

for-in循环

可以循环数组（列表）和对象（字典）

```
var o = {
      name: 'Jack',
      age: 20,
      city: 'Beijing'
};
for (var key in o) {
      console.log(key); // 'name', 'age', 'city'
}

var a = ['A', 'B', 'C'];
for (var i in a) {
      console.log(i); // '0', '1', '2'
      console.log(a[i]); // 'A', 'B', 'C'
}
```

while循环

条件循环

```
var x = 0;
var n = 99;
while (n > 0) {
      x = x + n;
      n = n - 2;
}
```

do-while循环

```
var n = 0;
do {
      n = n + 1;
} while (n < 100);
n; // 100
```

foreach循环

for-of循环

break语句

“跳出”循环。

```
for (i = 0; i < 10; i++) {
      if (i === 3) { break; }
      text += "数字是 " + i + "<br>";
}
```

continue语句

“跳过”循环中的一个迭代。

```
for (i = 0; i < 10; i++) {
      if (i === 3) { continue; }
      text += "数字是 " + i + "<br>";
}
```

3 条件选择

switch

Switch case 使用严格比较（===）。

```
switch(表达式) {
        case n:
             代码块
             break;
        case n:
             代码块
             break;
        default:
             默认代码块
}
```

第三章JavaScript函数

1 定义函数

函数

```
function abs(x) {
      if (x >= 0) {
            return x;
      } else {
            return -x;
      }
}

var abs = function (x) {
      if (x >= 0) {
            return x;
      } else {
            return -x;
      }
};

abs(10);
```

JavaScript的函数允许传入过多的参数或者过少的参数，也不会影响函数的执行。

函数的属性arguments

> 因为javascript中，函数也被看做对象，所以能给变量赋值。也能有函数的属性。

只在函数内部起作用，并且永远指向当前函数的调用者传入的所有参数

```
function foo(x) {
      console.log('x = ' + x); // 10
      for (var i=0; i<arguments.length; i++) {
            console.log('arg ' + i + ' = ' + arguments[i]); // 10, 20, 30
      }
}
foo(10, 20, 30);
```

函数的属性rest

用来表示可能的多余的参数

```
function foo(a, b, ...rest) {
      console.log('a = ' + a);
      console.log('b = ' + b);
      console.log(rest);
}

foo(1, 2, 3, 4, 5);
// 结果:
// a = 1
// b = 2
// Array [ 3, 4, 5 ]

foo(1);
// 结果:
// a = 1
// b = undefined
// Array []
```

2 变量作用域与解构赋值

作用范围

- 使用中括号确定一组作用域。只有函数会区分作用域。
- 作用域内部的变量，只在内部起作用。
- 不同作用域的相同变量相互独立。
- 作用域可以嵌套，外部作用域的变量可以在内部作用域使用，反之不可。
- 作用域可以嵌套，内部作用域的变量可以覆盖外部作用域的变量。

变量提升

JavaScript的函数定义有个特点，它会先扫描整个函数体的语句，把所有申明的变量“提升”到函数顶部。

也就是说，你在函数体中变量的声明被提前，但是变量的定义没有提前。

```
function a(){
      var x = 1;
      function b(){
            console.log(x);//x is undefined
            var x = 2;
      }
}
```

> 因为在函数b中，x的声明提前了，console输出的是，函数b中的x。但是x的定义还在后边，所以x的值是undefined。如果删掉内部的生命，则x是1.

全局作用域

不再任何函数体内定义的变量就是全局作用域。

- 默认全局变量window。
- 顶层寒素的定义也被是为一个全局变量，冰杯绑定到window对象。

```
function foo() {
      alert('foo');
}

foo(); // 直接调用foo()
window.foo(); // 通过window.foo()调用
```

名字空间

全局变量会绑定到window上，不同的JavaScript文件如果使用了相同的全局变量，或者定义了相同名字的顶层函数，都会造成命名冲突，并且很难被发现。

减少冲突的一个方法是把自己的所有变量和函数全部绑定到一个全局变量中。例如：

```
// 唯一的全局变量MYAPP:
var MYAPP = {};

// 其他变量:
MYAPP.name = 'myapp';
MYAPP.version = 1.0;

// 其他函数:
MYAPP.foo = function () {
      return 'foo';
};

```

把自己的代码全部放入唯一的名字空间MYAPP中，会大大减少全局变量冲突的可能。

局部作用域

javascript的变量作用域只有函数内部。在for循环语句块中无法定义具有局部作用域变量的。

可以使用let代替var申请块级作用域变量。

```
function foo() {
      var sum = 0;
      for (let i=0; i<100; i++) {
            sum += i;
      }
      // SyntaxError:
      i += 1;
}
```

解构赋值

```
//数组的结构赋值
var [x, y, z] = ['hello', 'JavaScript', 'ES6'];

//嵌套数组的结构赋值
```

```
let [x, [y, z]] = ['hello', ['JavaScript', 'ES6']];

//对象的解构赋值
var person = {
      name: '小明',
      age: 20,
      gender: 'male',
      passport: 'G-12345678',
      school: 'No.4 middle school',
      address: {
            city: 'Beijing',
            street: 'No.1 Road',
            zipcode: '100001'
      }
};
var {name, address: {city="he", zip}} = person;//使用默认赋值
name; // '小明'
city; // 'Beijing'
zip; // undefined, 因为属性名是zipcode而不是zip
// 注意: address不是变量，而是为了让city和zip获得嵌套的address对象的属性:
address; // Uncaught ReferenceError: address is not defined
```

3 变量的引用

- JavaScript中的变量都是引用。所以直接赋值，会将引用传递给另一个变量。应该通过创建新的类型，然后进行值的复制。

```
a={r:1};
b=a;
a.x=5;
b;//{r:1,x=5}
```

4 箭头函数

箭头函数的定义

```
x => x*x

(x, y) => x * x + y * y

// 无参数:
() => 3.14

// 可变参数:
(x, y, ...rest) => {
      var i, sum = x + y;
      for (i=0; i<rest.length; i++) {
            sum += rest[i];
      }
      return sum;
}
```

箭头函数的this

箭头函数内this指针指向当前词法作用域的对象。指向函数外层的对象。而且this指针无法更高绑定。

```
var obj = {
      birth: 1990,
      getAge: function () {
            var b = this.birth; // 1990
            var fn = () => new Date().getFullYear() - this.birth; // this指向obj对象
            return fn();
      }
};
obj.getAge();//25
```

6 generator函数

generator函数定义

```
function* foo(x) {
      yield x + 1;
      yield x + 2;
      return x + 3;
}
```

generator函数调用

```
 function* fib(max) {
       var
             t,
             a = 0,
             b = 1,
             n = 0;
       while (n < max) {
             yield a;
             [a, b] = [b, a + b];
             n ++;
       }
       return;
 }
```

直接调用调用，仅仅创建了一个generator对象，没有执行函数。不断调用next()方法，可以执行generator函数执行。每次遇到yield返回一次。

```
var f = fib(5);//一个generator对象
f.next(); // {value: 0, done: false}
f.next(); // {value: 1, done: false}
f.next(); // {value: 1, done: false}
f.next(); // {value: 2, done: false}
f.next(); // {value: 3, done: false}
f.next(); // {value: undefined, done: true}
```

利用for of方法访问generator对象。

```
for (var x of fib(10)) {
      console.log(x); // 依次输出0, 1, 1, 2, 3, ...
}
```

特点

因为generator可以在执行过程中多次返回，所以它看上去就像一个可以记住执行状态的函数，利用这一点，写一个generator就可以实现需要用面向对象才能实现的功能。

generator还有另一个巨大的好处，就是把异步回调代码变成“同步”代码。

简单来说，就是一个局部状态变量。

7 装饰器

> (java中称为注解，Python也称为装饰器)

因为JavaScript是脚本语言所有的对象、变量都是动态的。可以随时解绑。可以利用重新绑定的方法，修改原来的函数，为原来的函数添加装饰器。

```
//统计函数的执行次数
var count = 0;
var oldParseInt = parseInt; // 保存原函数

window.parseInt = function () {
      count += 1;
      return oldParseInt.apply(null, arguments); // 调用原函数
};
```

第四章JavaScript高阶函数

1 高阶函数定义

JavaScript的函数其实都指向某个变量。既然变量可以指向函数，函数的参数能接收变量，那么一个函数就可以接收另一个函数作为参数，这种函数就称之为高阶函数。

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

x = -5;
y = 6;
f = Math.abs;
f(x) + f(y) ==> Math.abs(-5) + Math.abs(6) ==> 11;
return 11;
```

2 MapReduce

map:对集合分割运算。
reduce：对分割运算的结果进行聚合。

array.map

map()方法本身可以接受一个函数，作用在数组的每一个元素，并拼接成新的数组。是一个高阶函数。

```
function pow(x) {
      return x * x;
}
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
var results = arr.map(pow); // [1, 4, 9, 16, 25, 36, 49, 64, 81]
console.log(results);

var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
arr.map(String); // ['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

array.reduce

再看reduce的用法。Array的reduce()把一个函数作用在这个Array的[x1, x2, x3...]上，这个函数必须接收两个参数，reduce()把结果继续和序列的下一个元素做累积计算，其效果就是：

```
[x1, x2, x3, x4].reduce(f) = f(f(f(x1, x2), x3), x4)
```

```
var arr = [1, 3, 5, 7, 9];
arr.reduce(function (x, y) {
      return x + y;
}); // 25
```

3 filter

过滤元素

把Array的某些元素过滤掉，然后返回剩下的元素

Array的filter()也接收一个函数。filter()把传入的函数依次作用于每个元素，然后根据返回值是true还是false决定保留还是丢弃该元素。

```
var arr = [1, 2, 4, 5, 6, 9, 10, 15];
var r = arr.filter(function (x) {
      return x % 2 !== 0;
});
r; // [1, 5, 9, 15]
```

回调函数

filter()接收的回调函数，其实可以有多个参数。通常我们仅使用第一个参数，表示Array的某个元素。回调函数还可以接收另外两个参数，表示元素的位置和数组本身：

```
var arr = ['A', 'B', 'C'];
var r = arr.filter(function (element, index, self) {
      console.log(element); // 依次打印'A', 'B', 'C'
      console.log(index); // 依次打印0, 1, 2
      console.log(self); // self就是变量arr
      return true;
});
```

利用回调函数，除掉重复元素

```
var r,arr = ['apple', 'strawberry', 'banana', 'pear', 'apple', 'orange', 'orange', 'strawberry'];
r = arr.filter(function (element, index, self) {
      return self.indexOf(element) === index;
});

```

4 sort

默认把所有的元素转换为String进行排序。可以接受比较函数进行排序。

```
 arr.sort(function (x, y) {
       if (x < y) {
             return -1;
       }
       if (x > y) {
             return 1;
       }
       return 0;
 });
 console.log(arr); // [1, 2, 10, 20]
```

5 every

every()方法可以判断数组的所有元素是否满足测试条件。

```
var arr = ['Apple', 'pear', 'orange'];
console.log(arr.every(function (s) {
      return s.length > 0;
})); // true, 因为每个元素都满足s.length>0

console.log(arr.every(function (s) {
      return s.toLowerCase() === s;
})); // false, 因为不是每个元素都全部是小写
```

6 find

find()方法用于查找符合条件的第一个元素，如果找到了，返回这个元素，否则，返回undefined

```
var arr = ['Apple', 'pear', 'orange'];
console.log(arr.find(function (s) {
      return s.toLowerCase() === s;
})); // 'pear', 因为pear全部是小写

console.log(arr.find(function (s) {
      return s.toUpperCase() === s;
})); // undefined, 因为没有全部是大写的元素
```

7 findIndex

findIndex()和find()类似，也是查找符合条件的第一个元素，不同之处在于findIndex()会返回这个元素的索引，如果没有找到，返回-1：

```
var arr = ['Apple', 'pear', 'orange'];
console.log(arr.findIndex(function (s) {
      return s.toLowerCase() === s;
})); // 1, 因为'pear'的索引是1

console.log(arr.findIndex(function (s) {
      return s.toUpperCase() === s;
})); // -1
```

8 forEach

forEach()和map()类似，它也把每个元素依次作用于传入的函数，但不会返回新的数组。forEach()常用于遍历数组，因此，传入的函数不需要返回值：

```
var arr = ['Apple', 'pear', 'orange'];
arr.forEach(console.log); // 依次打印每个元素
```

9 闭包

> 闭包的出现，可以获取函数内部的变量且变量一直在内存当中.最简单的理解，一个作用域使用了另一个作用域中的变量。比如外层函数通过传参的方式，把某个变量传递给内层函数，就产生了闭包

函数可以作为返回值实现惰性求值

高阶函数除了可以接受函数作为参数外，还可以把函数作为结果值返回。

可以通过函数作为返回值，实现懒求值。

```
function lazy_sum(arr) {
      var sum = function () {
            return arr.reduce(function (x, y) {
                  return x + y;
            });
      }
      return sum;
}

//调用函数f时，才真正计算求和的结果：

f(); // 15
```

函数作为返回值实现循环惰性求值

> 感觉与Python的生成器yield惰性求值很像，这里介绍了其根本的原理。

```
function count() {
      var arr = [];
      for (var i=1; i<=3; i++) {
            arr.push((function (n) {
                  return function () {
                        return n * n;
                  }
            })(i));
      }
      return arr;
}

var results = count();
var f1 = results[0];
var f2 = results[1];
var f3 = results[2];

f1(); // 1
f2(); // 4
f3(); // 9
```

10 回调函数与多线程异步通信

回调函数就是一个函数，它是在我们启动一个异步任务的时候就告诉它：等你完成了这个任务之后要干什么。这样一来主线程几乎不用关心异步任务的状态了，他自己会善始善终。

第五章JavaScript异常处理

异常处理语句

try 语句使您能够测试代码块中的错误。

catch 语句允许您处理错误。

throw 语句允许您创建自定义错误。

finally 使您能够执行代码，在 try 和 catch 之后，无论结果如何。

```
try {
        供测试的代码块
}
 catch(err) {
        处理错误的代码块
}
finally {
        无论 try / catch 结果如何都执行的代码块
}
```

第六章JavaScript对象

1 对象定义

- 在JavaScript中所有事物都是对象。
- 对象也是一个变量，可以包含任意值。 `var car = {type:"Fiat", model:500, color:"white"};`

2 对象属性

定义

- 键值对

访问

- person.lastName
- person["lastName"]

3 对象方法

对象绑定的函数称为方法

```
var xiaoming = {
      name: '小明',
      birth: 1990,
      age: function () {
            var y = new Date().getFullYear();
            return y - this.birth;
      }
};

xiaoming.age; // function xiaoming.age()
xiaoming.age(); // 今年调用是25,明年调用就变成26了
```

this变量

- this指针一直指向当前的对象。
- 如果是全局对象，this指向window
- this指向的具体位置，视调用者不同而改变。

```
function getAge() {
      var y = new Date().getFullYear();
      return y - this.birth;
}

var xiaoming = {
      name: '小明',
      birth: 1990,
      age: getAge
};

xiaoming.age(); // 25, 正常结果，this指向小明。
getAge(); // NaN,this指向window
```

- 对象的方法中的嵌套方法，this指向不明确（window）。所以，如果在子作用域调用本作用域的内容，定义that进行指针传递。

```
var xiaoming = {
      name: '小明',
      birth: 1990,
      age: function () {
            var that = this; // 在方法内部一开始就捕获this
            function getAgeFromBirth() {
                  var y = new Date().getFullYear();
                  return y - that.birth; // 用that而不是this
            }
            return getAgeFromBirth();
      }
};

xiaoming.age(); // 25
```

apply()

使用函数的apply属性，重新定义this指针的指向。

即，将函数应用到某个对象上。

函数本身的apply方法，它接收两个参数，第一个参数就是需要绑定的this变量，第二个参数是Array，表示函数本身的参数。

```
function getAge() {
      var y = new Date().getFullYear();
      return y - this.birth;
}

var xiaoming = {
      name: '小明',
      birth: 1990,
      age: getAge
};

xiaoming.age(); // 25
getAge.apply(xiaoming, []); // 25, this指向xiaoming, 参数为空
```

call()

使用函数的call属性，同样可以定义this指针的具体指向。

与apply()类似的方法，唯一区别是apply()把参数打包成Array再传入；call()把参数按顺序传入。

比如调用Math.max(3, 5, 4)，分别用apply()和call()实现如下：

```
Math.max.apply(null, [3, 5, 4]); // 5
Math.max.call(null, 3, 5, 4); // 5
```

4 new运算符

实例

new 运算符创建一个用户定义的对象类型的实例或具有构造函数的内置对象的实例。

```
function Car(make, model, year) {
   this.make = make;
   this.model = model;
   this.year = year;
}

const car1 = new Car('Eagle', 'Talon TSi', 1993);

console.log(car1.make);
// expected output: "Eagle"
```

语法

```
new constructor([arguments])
```

原理

1. 创建一个空的简单JavaScript对象（即{}）；
2. 链接该对象（即设置该对象的构造函数）到另一个对象；
3. 将步骤1新创建的对象作为this的上下文；
4. 如果该函数没有返回对象，则返回this。

步骤

创建一个用户自定义的对象需要两步：

1. 通过编写函数来定义对象类型。
2. 通过 new 来创建对象实例。

第七章JavaScript对象分类

0 对象

typeof

在JavaScript一切皆对象。

使用typeof运算符获取对象类型。返回对象类型。

包装对象

将基本类型转换为对象类型。

```
var n = new Number(123); // 123,生成了新的包装类型
var b = new Boolean(true); // true,生成了新的包装类型
var s = new String('str'); // 'str',生成了新的包装类型
```

1 JavaScript内建对象

Array对象

Array 对象用于在变量中存储多个值

Boolean对象

Boolean 对象用于转换一个不是 Boolean 类型的值转换为 Boolean 类型值 (true 或者false).

Date对象

Date 对象用于处理日期与时间。

Math对象

Math 对象用于执行数学任务。

Number对象

原始数值的包装对象

String对象

处理文本字符串

RegExp对象

正则表达式字符串的处理

全局属性、函数

encode、decode系列
数据类型转换系列

Error对象

在错误发生时，提供了错误的提示信息。

2 Browser对象

window对象

表示浏览器打开的窗口。是默认的全局对象。默认绑定了alert等方法。

Navigator对象

包含浏览器相关的信息。

Screen对象

客户端显示屏相关的信息。

History对象

包含用户在浏览器中访问过的URL。是window对象的一部分。

Location对象

包含当前的URL信息。是window对象的一部分。

存储对象

webAPI提供了sessionStorage(会话存储)和localStorage(本地存储)两个对象来对网页的数据进行增删查改操作。

localStorage 用于长久保存整个网站的数据，保存的数据没有过期时间，直到手动去除。

sessionStorage 用于临时保存同一窗口(或标签页)的数据，在关闭窗口或标签页之后将会删除这些数据。

3 DOM对象

- 文档是一个文档节点
- 所有的HTML元素是元素节点
- 所有的HTML属性是属性节点
- 文本插入到HTML元素是文本节点
- 注释是注释节点。

DOM Document对象

当浏览器载入 HTML 文档, 它就会成为 Document 对象。

Document 对象是 HTML 文档的根节点。

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问

DOM元素对象

元素对象代表着一个 HTML 元素。

元素对象 的 子节点可以是, 可以是元素节点，文本节点，注释节点。

NodeList 对象 代表了节点列表，类似于 HTML元素的子节点集合。

元素可以有属性。属性属于属性节点（查看下一章节）。

DOM属性对象

在 HTML DOM 中, Attr 对象 代表一个 HTML 属性。

HTML属性总是属于HTML元素。

在 HTML DOM 中, the NamedNodeMap 对象 表示一个无顺序的节点列表。

我们可通过节点名称来访问 NamedNodeMap 中的节点。

DOM事件对象

HTML DOM 事件允许Javascript在HTML文档元素中注册不同事件处理程序。

事件通常与函数结合使用，函数不会在事件发生前被执行！（如用户点击按钮）。

DOMConsole对象

Console 对象提供了访问浏览器调试模式的信息到控制台。

CSS Style Declaration对象

CSSStyleDeclaration 对象表示一个 CSS 属性-值（property-value）对的集合。

DOMHTMLCollection

HTMLCollection 是 HTML 元素的集合。

HTMLCollection 对象类似一个包含 HTML 元素的数组列表。

第八章JavaScript事件响应

1 常见事件

> 本质上也是一种异步通信的方式。基于回调的异步通信，和基于事件响应机制的异步通信。

鼠标事件

- onclick 用户点击了 HTML 元素
- ondblclick 当用户双击某个对象时调用的事件句柄
- onmousedown 鼠标按钮被按下。
- onmouseenter 当鼠标指针移动到元素上时触发。
- onmouseleave 当鼠标指针移出元素时触发
- onmousemove 鼠标被移动。
- onmouseover 鼠标移到某元素之上。
- onmouseout 鼠标从某元素移开。
- onmouseup 鼠标按键被松开。

键盘事件

- onkeydown 用户按下键盘按键
- onkeypress 某个键盘按键被按下并松开。
- onkeyup 某个键盘按键被松开。

框架对象事件

- onabort 图像的加载被中断。（）2
- onbeforeunload 该事件在即将离开页面（刷新或关闭）时触发 2
- onerror 在加载文档或图像时发生错误。（,和）
- onhashchange 该事件在当前 URL 的锚部分发生修改时触发。
- onload 一张页面或一幅图像完成加载。 2
- onpageshow 该事件在用户访问页面时触发
- onpagehide 该事件在用户离开当前网页跳转到另外一个页面时触发
- onresize 窗口或框架被重新调整大小。 2
- onscroll 当文档被滚动时发生的事件。 2
- onunload 用户退出页面。（ 和）

表单事件

- onblur 元素失去焦点时触发 2
- onchange 该事件在表单元素的内容改变时触发(,,

)

```
<button onclick="displayDate()">点这里</button>
```

通过JS给HTML对象添加事件属性，并绑定方法

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

第九章JavaScriptJSON

1 JSON 简介

JSON 是存储和传输数据的格式。

JSON 经常在数据从服务器发送到网页时使用。

- JSON 指的是 JavaScript Object Notation
- JSON 是轻量级的数据交换格式
- JSON 独立于语言 *
- JSON 是“自描述的”且易于理解

2 JSON使用

基本语法

- 数据是名称/值对
- 数据由逗号分隔
- 花括号保存对象
- 方括号保存数组

JSONObject

JSON 数据的书写方式是名称/值对。

名称/值对由（双引号中的）字段名构成，其后是冒号，再其后是值：
JSON 对象是在花括号内书写的。

```
{"firstName":"Bill", "lastName":"Gates"}
```

JSONArray

JSON 数组在方括号中书写。

```
"employees":[
    {"firstName":"Bill", "lastName":"Gates"},
    {"firstName":"Steve", "lastName":"Jobs"},
    {"firstName":"Alan", "lastName":"Turing"}
]
```

JSONString

```
var text = '{ "employees" : [' +
'{ "firstName":"Bill" , "lastName":"Gates" },' +
'{ "firstName":"Steve" , "lastName":"Jobs" },' +
'{ "firstName":"Alan" , "lastName":"Turing" } ]}';
```

JSONString转换为JavaScript对象

```
var obj = JSON.parse(text);
```

JavaScript对象转换为JSONString

```
var myJSON = JSON.stringify(obj);
```

第十章JavaScript原型链

1 函数与对象的表面关系

- 函数是函数，对象是对象。
- 函数本身也是特殊对象，typeof 返回function。函数返回值可以是对象。
- 对象可以包含函数属性。typeof 返回object
- 函数可以作为对象的模板，称为对象构造器，typeof返回值，返回object。常见的内置类型构造器，首字母大写，通过new关键字，返回新的对象。new Array（）；new Function();

- 变量可以指向函数，也可以指向对象。
- 函数的定义方法 function(){}匿名函数；或者function name(){}
- 对象的定义方法 new Creator();或者{}。从对象构造器中创建对象，或者直接创建对象。通过new创建的对象，proto指向Creator()对象构造器。通过{}创建的对象，proto指向Object()对象构造起，会形成，proto链。

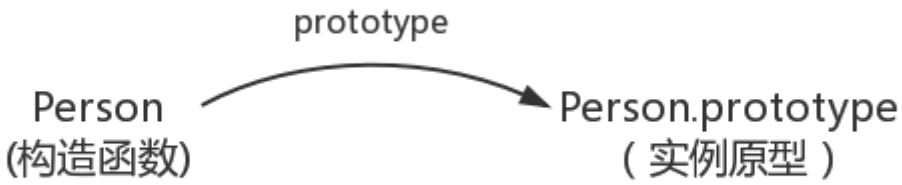
2 原型

函数原型

- 原型的概念：每一个javascript对象(除null外)创建的时候，就会与之关联另一个对象，这个对象就是我们所说的原型，每一个对象都会从原型中“继承”属性。

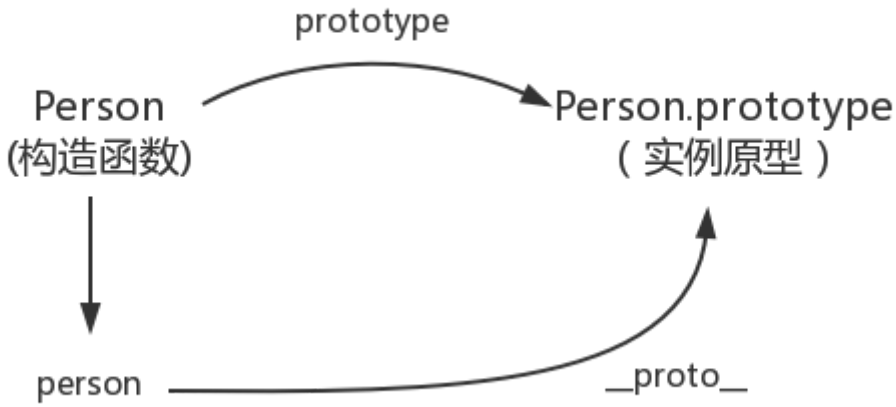
```
function Person(age) {
    this.age = age
}
Person.prototype.name = 'kavin'
var person1 = new Person()
var person2 = new Person()
console.log(person1.name) //kavin
console.log(person2.name)  //kavin
```

- 在JavaScript中，每个函数都有一个prototype属性，这个属性指向函数的原型对象。函数的prototype指向了一个对象，而这个对象正是调用构造函数时创建的实例的原型，也就是person1和person2的原型。



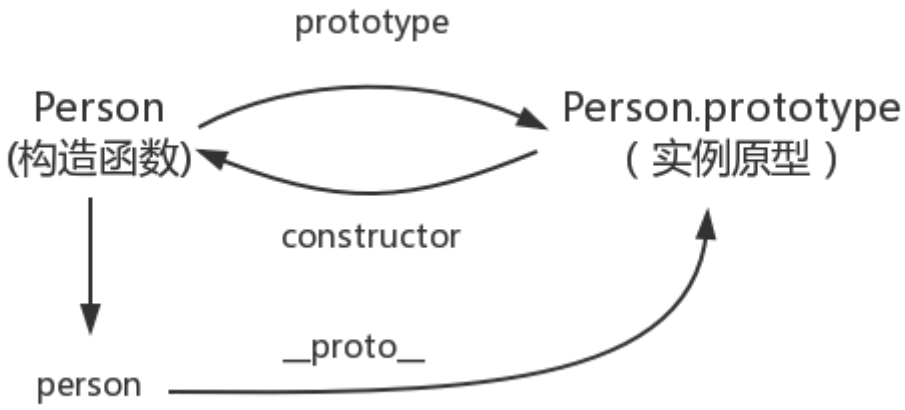
对象原型

- 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。



construtor构造函数

- 每个原型都有一个constructor属性，指向该关联的构造函数。



实例与原型

- 当读取实例的属性时，如果找不到，就会查找与对象关联的原型中的属性，如果还查不到，就去找原型的原型，一直找到最顶层为止。

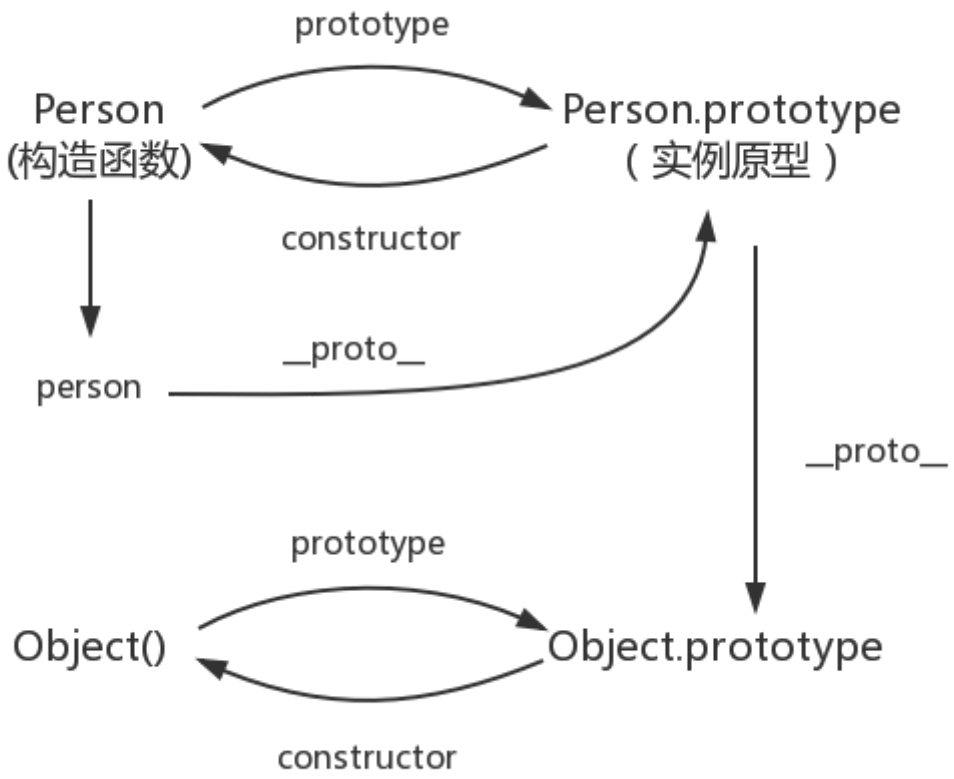
```
function Person() {

}

Person.prototype.name = 'Kevin';

var person = new Person();

person.name = 'Daisy';
console.log(person.name) // Daisy

delete person.name;
console.log(person.name) // Kevin
```

原型的原型

原型也是一个对象，既然是对象，我们就可以用最原始的方式创建它，那就是：

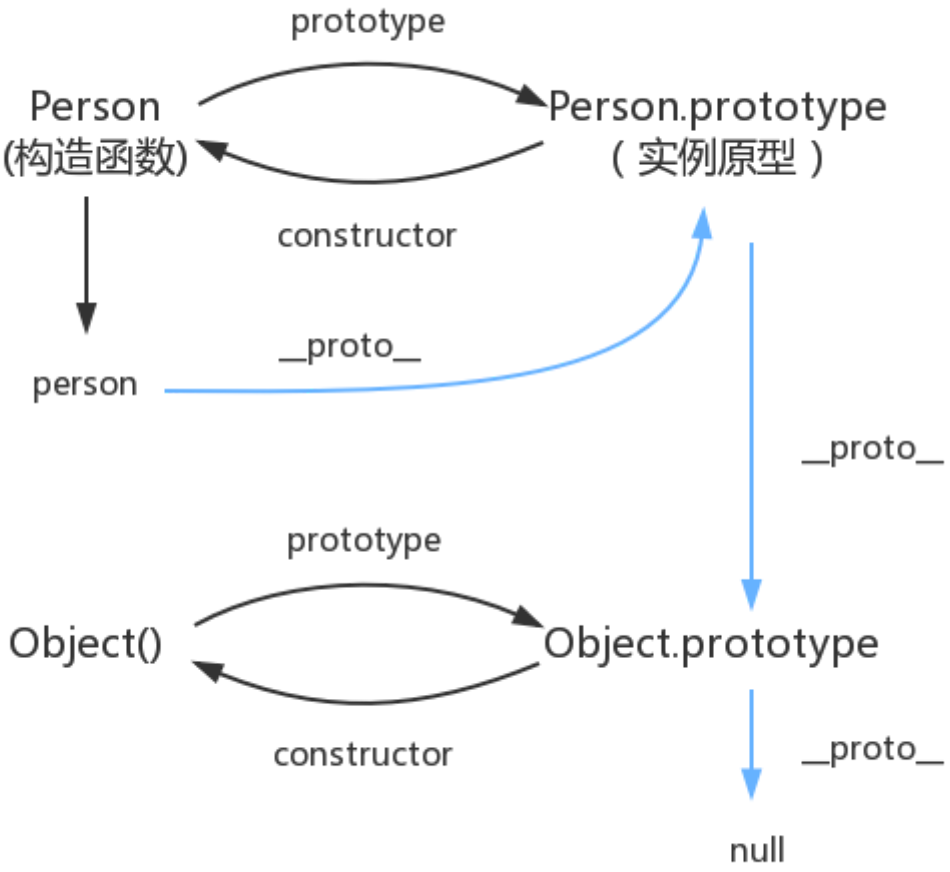
```
var obj = new Object();
obj.name = 'Kevin'
console.log(obj.name) // Kevin
```

其实原型对象就是通过 Object 构造函数生成的，结合之前所讲，实例的 **proto** 指向构造函数的 prototype .



原型链

- 简单的回顾一下构造函数、原型和实例的关系：每个构造函数都有一个原型对象，原型对象都包含一个指向构造函数的指针，而实例都包含一个指向原型对象的内部指针。那么假如我们让原型对象等于另一个类型的实例，结果会怎样？显然，此时的原型对象将包含一个指向另一个原型的指针，相应地，另一个原型中也包含着一个指向另一个构造函数的指针。假如另一个原型又是另一个类型的实例，那么上述关系依然成立。如此层层递进，就构成了实例与原型的链条。这就是所谓的原型链的基本概念。



补充

- 函数的原型属性，指向原型对象。prototype属性。
- 对象的原型属性，指向原型对象。**proto**属性。
- 原型对象是实例的原型。
- 原型对象的constructor指向对象构造器，也就是函数。
- 原型对象通过constructor和对象构造器通过prototype构成了指向环。
- 原型对象本身也是实例，她本身也具有原型对象。通过原型对象的原型对象，形成了原型链。原型对象通过**prpto**属性指向自己的原型。

> 也就是说，原型对象，有两个属性，一个是构造器constructor，指向对象构造器。另一个属性指向自身的原型对象。普通对象没有constructor属性。

第十一章BOM

1 window对象

当前的窗口对象

全局所有的对象的父对象

2 Navigator对象

当前的浏览器对象

浏览器的名称、版本、代理等信息

3 Screen对象

屏幕对象

屏幕的尺寸

4 location 对象

url对象

能够用来解析url，访问新的地址，请求的协议和端口号。

5 storage对象

1. 本地存储对象localStorage 将数据存储到本地
2. sessionStorage对象，将数据存储为会话数据。

6 document对象

整个HTML页面的对象。

能够得到所有的文档节点、元素节点、属性节点、文本节点、注释节点

1. 能够得到当前页面包含的cookie。保存当前的登录信息

7 history对象

代表浏览器的历史记录。

1. history.back
2. history.forward

第十二章DOM

DOM可以在w3school离线开发手册里查看

查看方法如下

1.双击JavaScript

2.点击左边课程表的HTML DOM

3.点击左边的DOM参考