`

JavaScript 语言精粹二

对于丑陋的事物，爱会闭目无视。 - 威廉·莎士比亚《维洛那二绅士》

起步：

- 本章节引用莎士比亚名句作为开端，对 JavaScript 重点'对象'进行解析。
- JavaScript 的简单数据类型包括数字、字符串、布尔值、null 和 undefined 值，其他值为对象。
  - 数字、字符串和布尔值貌似为对象，因为它们具有方法，但它们是不可变的。
- JavaScript 中对象是可变的键控集合，在 JS 中数组、函数、正则当然也包括对象它们都是对象。
  - 对象是属性的容器，每个属性都拥有键名和值，属性键名可以是包括空字符串在内的任意字符串，值也可以是除undefined以外的任意值。
  - JavaScript里的对象是无类型的（class-fee），它对新属性的名字和值没有限制。
  - 对象适用于汇集和管理数据，对象也可以包括对象，所以它更容易形成树状或图形结构。
  - JavaScript包含一种原型链的特性，允许对象继承另一个对象的属性，正确的使用可减少对象初始化消耗的时间及内存

对象字面量

- 对象字面量提供一种非常方便地创建新对象值得表示法。
  - 包围在一对花括号中的零或多个'键/值'对。

如下：

```
let empty_obj = {};

let obj = {
  name_line: "Pooo",
  "name-cut": "Pooo",
  gender: "boy",
};
```

- 属性名可以是包含空字符串在内的任意字符串。在对象字面量中，若属性名合法标识符（非保留字），则不强制要求用引号括住属性名，所以上文 name_line 下划线标准写法中引号可以省略，而 name-cut 减号的""必须有。
- 并且，对象可嵌套，如下：

```
var father = {
  name: "Potato",
  age: "22",
  son: {
    first: "Tomato",
    second: "cucumber",
  },
};
```

检索

- 要检索对象里包含的值，可用以[ ]表达式，若字符串是合法标识符而非保留字，则也可用.调用，常规下，优先使用紧凑且可读性好的.调用。

```
father.name; //"Potato"
father["name"]; //"Potato"
```

- 若尝试检索不存在的成员属性值，则返回undefined

```
father.grandson; //undefined
father["grandson"]; //undefined
```

- 利用运算符||可以进行默认值填充：

```
let dear = father.daughter || "none"; //none
let gender = father.gender; //undefined
```

> 注意：在 JavaScript 中连接符（ - ）通常是不合法的，但允许使用下划线（ _ ）

更新

- 对象中的值可以通过赋值来进行更新，若属性名已经存在，那么这个属性会被覆盖掉。

```
var father = {
  name: "Potato",
  age: "22",
  son: {
    first: "Tomato",
    second: "cucumber",
  },
};
console.log(father.age); //22
father.age = 23;
console.log(father.age); //23
```

- 若该对象之前没有拥有该属性名，则会被扩充到此对象中。

```
  father.city='HangZhou';
{name: "Potato", age: 23, son: {…}, city: "HangZhou"}
    age: 23
```

```
    city: "HangZhou"
    name: "Potato"
    son: {first: "Tomato", second: "cucumber"}
```

引用

- 对象可通过引用来传递，但它们永远不会被复制
  - 无论 refBox 还是 getBox 它们都指向同一个对象引用，因此共享属性

```
  var box={
    check:'book',
  }
  var refBox=box;
  box.price='25元';
  var getBox=box.price;//'25元'
```

- x、y、z 它们每个都引用了不同的对象，因此也各不影响

```
var x = {},
  y = {},
  z = {};
x == y; //false
var a = (b = c = {});
a === b; //true
```

原型

- 每个对象都关联到一个原型对象，并可从中继承相应属性。所有通过对象字面量创建的对象都连接到 Object.prototype,它也是 JavaScript 中的终点对象。
- 当创建一个新对象时，可以选择某个对象为其原型，可以尝试给 Object 添加一个 create 方法，它可以创建一个使用**原对象**作为其**原型**的新对象。

```
  var box={
    check:'book',
    checkbox:{
      check:'pencil',
      price:'30元'
    },
    getPrice:function(){
      return '25元'
    }
  }
if(typeof Object.beget!=='function'){
  Object.create=function(o){
    var F=function(){};
    F.prototype=o;//使其原型改为传进来的对象
```

```
    return new F();
  }
}
var goodsInfo=Object.create(box)
console.log(goodsInfo.check)//'book'
console.log(goodsInfo.getPrice())//'25元'
```

> 注意：原型链只有检索值时才用到，若尝试回去对象中某属性值时，它不存在此属性名，则javascript会试着从原型对象中获取属性值，若原型对象中也没有，则从原型中寻找，以此类推至Object.prototype,无则返回undefined。

反射

- 检查并确定对象具有的属性，只有进行检索并验证即可，通常使用typeof

```
typeof box.check; //"string"
typeof box.checkbox; //"object"
typeof box.getPrice; //"function"
```

- 原型链
- 但原型链中的任何属性还会产生同一个值constructor，例如：

```
typeof box.toString; //'function'
typeof box.constructor; //'function'
```

- 为了处理掉这些不需要的属性，可以换种方法来验证，比如利用hasOwnProperty
- 它用来检验是否是对象独有的属性，它将返回对应的布尔值，它不会检查原型链

```
box.hasOwnProperty("check"); //true
box.hasOwnProperty("getcheck"); //false
```

枚举

- for in 语句用来遍历对象中的所有属性名。该枚举过程会列出所有的属性（包括函数和原型链中的属性）
- 同时一般利用 typeof 来排除不想要的函数

```
var box = {
  check: "book",
  checkbox: {
    check: "pencil",
    price: "30元",
  },
  getPrice: function () {
    return "25元";
```

```
  },
};

for (name in box) {
  if (typeof box[name] !== "function") {
    console.log(name + ":" + box[name]);
    //check:book
    //checkbox:[object Object]
  }
}
```

- 因为属性名出现的顺序具有不确定性，因此要对可能出现的顺序有所准备
- 若想避免这种不确定性出现，那么应避免出现for in语句，转而换为数组循环

```
var box = ["book", "pencil", "30元"];

for (let i = 0; i < box.length; i++) {
  console.log(box[i]);
}
```

删除

- delete 运算符可以用来删除某对象的属性，若此对象含有该属性，则移除（不会对原型链中的对象产生影响）
- 删除某对象的属性，可能会让它原型链中的此属性表现出来
- 若原型链 object 上有 a 属性，而对象 A 身上也有 a 属性，则 A 使用时以自身为准
- （这也是为什么，删除 A 身上的 a 属性时，A 身上的 object 的 a 属性生效的原因）如下：

```
function Pro() {}
Pro.prototype = {
  constructor: Pro,
  name: "proto",
  sex: "boy",
  hobby: "coding",
};
var fakePro = new Pro();
fakePro.hobby = "钓鱼🎧";
console.log(fakePro.hobby); //'钓鱼🎧'
delete fakePro.hobby;
console.log(fakePro.hobby); //'coding'
```

减少全局变量污染

- JavaScript可以随意定义全局变量（当然，这在方便的同时带来很多隐患）
- 为降低这些隐患，一般为其应用创建唯一全局变量容器
- 把全局资源都归纳至一个名称空间下，则会显著降低命名冲突。

```
var box = {};
box.content = {
  first_box: "tomato",
  second_box: "cucumber",
  third_box: "potato",
};
box.price = {
  tomato: "25",
  cucumber: "15",
  potato: "10",
};
```

总结：

> 本章节描述的是在JavaScript语言中的对象的基本概念，更加多元化的使用方法及重要知识点将在函数章节中记录。