ES5/String

严格模式（了解）

- 我们都知道 js 是一个相对不很严谨的语言
- 而且开发的时候，一些代码也不是很严格要求
- 而严格模式就是对开发的时候写的一些内容做了要求

开启严格模式

- 想开启严格模式，直接在代码最开始的位置写上字符串 `use strict`

```
<script>
 'use strtic'
 // 下面代码书写就要按照严格模式来书写
</script>
```

严格模式的规则

1. 声明变量必须有 `var` 关键字

```
'use strtic'

var num = 100
num2 = 200 // 这个就会报错
```

   - 之前了解过，在声明变量的时候，如果没有 var 关键字，那么按照作用域的规则会自动定义成全局变量
   - 严格模式下不可以，会报错

2. 函数的行参不可以重复

```
'use strtic'

function fn(p1, p1) {} // 直接就会报错
```

   - 在非严格模式下，函数两个行参一样，是不会报错的，只不过就是相当于在函数内部只有一个变量了
   - 但是在严格模式下会报错

3. 声明式函数调用的时候函数内部没有 this

```
'use strtic'

function fn() {
 console.log(this) // undefined
}
fn()
```

- 本身，全局声明式函数在调用的时候，函数内部的 this 是指向 window 的
- 在严格模式下，是没有 this 的

ES5 中常见的数组常用方法

- 之前我们讲过的数组常用方法都是 ES3 的方法
- 今天来说一些 ES5 中的方法

indexOf

- `indexOf` 用来找到数组中某一项的索引
- 语法： `indexOf(你要找的数组中的项)`

```
var arr = [1, 2, 3, 4, 5]

// 使用 indexOf 超找数组中的某一项
var index = arr.indexOf(3)

console.log(index) // 2
```

- 我们要找的是数组中值为 3 的那一项
- 返回的就是值为 3 的那一项在该数组中的索引

- 如果你要找的内容在数组中没有，那么就会返回 -1

```
var arr = [1, 2, 3, 4, 5]

// 使用 indexOf 超找数组中的某一项
var index = arr.indexOf(10)

console.log(index) // -1
```

- 你要找的值在数组中不存在，那么就会返回 -1

forEach

- 和 for 循环一个作用，就是用来遍历数组的
- 语法：`arr.forEach(function (item, index, arr) {})`

```
var arr = [1, 2, 3]

// 使用 forEach 遍历数组
arr.forEach(function (item, index, arr) {
  // item 就是数组中的每一项
  // index 就是数组的索引
  // arr 就是原始数组
  console.log('数组的第 ' + index + ' 项的值是 ' + item + ', 原始数组是', arr)
})
```

  - forEach() 的时候传递的那个函数，会根据数组的长度执行
  - 数组的长度是多少，这个函数就会执行多少回

map

- 和 forEach 类似，只不过可以对数组中的每一项进行操作，返回一个新的数组

```
var arr = [1, 2, 3]

// 使用 map 遍历数组
var newArr = arr.map(function (item, index, arr) {
  // item 就是数组中的每一项
  // index 就是数组的索引
  // arr 就是原始数组
  return item + 10
})

console.log(newArr) // [11, 12, 13]
```

filter

- 和 map 的使用方式类似，按照我们的条件来筛选数组
- 把原始数组中满足条件的筛选出来，组成一个新的数组返回

```
var arr = [1, 2, 3]

// 使用 filter 过滤数组
var newArr = arr.filter(function (item, index, arr) {
```

```
  // item 就是数组中的每一项
  // index 就是数组的索引
  // arr 就是原始数组
  return item > 1
 })

 console.log(newArr) // [2, 3]
```

- 我们设置的条件就是 `> 1`
- 返回的新数组就会是原始数组中所有 `> 1` 的项

创建字符串（了解）

- 我们创建字符串也分为两种方法 **字面量** 和 **构造函数**
- 字面量：

```
var str = 'hello'
```

- 构造函数创建

```
var str = new String('hello')
```

ASCII 字符集（了解）

- 我们都知道，计算机只能存储 `0101010` 这样的二进制数字
- 那么我们的 `a ~ z` / `A ~ Z` / `$` / `@` /… 之类的内容也有由二进制数字组成的
- 我们可以简单的理解为， `a ~ z` / `A ~ Z` / `$` / `@` /… 之类的内容都有一个自己的编号，然后在计算机存储的时候，是存储的这些编号，我们看的时候，也是通过这些编号在解析成我们要看到的内容给我们看到

- 上面的就是 ASCII 对照表，我们只需要知道他是这么存储的就好

unicode 编码

- 我们看到了，ASCII 只有这 128 个字符的编码结构
- 但是因为 ASCII 出现的比较早，而且是美国发明的，早起时候这些内容就够用了
- 因为存储一些英文的内容，传递一些英文的文章什么的都够用了
- 那么对于这个世界来说肯定是不够用的

- 因为我们的汉字没有办法存储，包括一些其他国家的语言也没有办法存储
- 所以就出现了 unicode 编码，也叫（万国码，统一码）
- unicode 对照表就是一个和 ASCII 一样的对照表，只不过变得很大很大，因为存储的内容特别的多
- 而且包含了世界上大部分国家的文字，所以我们的文字和字符现在在存储的时候，都是按照 unicode 编码转换成数字进行存储
- 我们的 UTF-8 就是一种 8 位的unicode字符集

字符串的常用方法

- 我们操作字符串，也有一堆的方法来帮助我们操作
- 字符串和数组有一个一样的地方，也是按照索引来排列的

charAt

- `charAt(索引)` 是找到字符串中指定索引位置的内容返回

```
var str = 'Jack'

// 使用 charAt 找到字符串中的某一个内容
var index = str.charAt(2)

console.log(index) // c
```

  - 因为字符串也是按照索引进行排列的，也是同样从 0 开始
  - 所以索引 2 的位置就是 c

- 如果没有对应的索引，那么就会返回 空字符串

```
var str = 'Jack'

// 使用 charAt 找到字符串中的某一个内容
var index = str.charAt(10)

console.log(index) // ''
```

  - 这个字符串根本没有索引 10 的位置
  - 所以就会返回一个空字符串 `''`

charCodeAt

- `charCodeAt(索引)` 就是返回对应索引位置的 unicode 编码

```
var str = 'Jack'
```

```
2.
3.  // 使用 charAt 找到字符串中的某一个内容
4.  var index = str.charCodeAt(0)
5.
6.  console.log(index) // 74
```

- 因为 `J` 在 unicode 对照表里面存储的是 74，所以就会返回 74

indexOf

- `indexOf` 就是按照字符找到对应的索引

```
1.  var str = 'Jack'
2.
3.  // 使用 indexOf 找到对应的索引
4.  var index = str.indexOf('J')
5.
6.  console.log(index) // 0
```

- 因为字符 `J` 在字符串 `Jack` 中的索引位置是 0
- 所以会返回 0

substring

- `substring` 是用来截取字符串使用的
- 语法：`substring(从哪个索引开始，到哪个索引截止)`，包含开始索引，不包含结束索引

```
1.  var str = 'hello'
2.  // 01234
3.
4.  // 使用 substring 截取字符串
5.  var newStr = str.substring(1, 3)
6.
7.  console.log(newStr) // el
```

- 从索引 1 开始，到索引 3 截止，包含前面的索引不包含后面的索引
- 所以返回的是 el

substr

- `substr` 也是用来截取字符串的
- 语法：`substr(从哪个索引开始，截取多少个)`

```
1.  var str = 'hello'
```

```
var str = 'hello'
// 01234

// 使用 substr 截取字符串
var newStr = str.substr(1, 3)

console.log(newStr) // ell
```

- 这个方法和 `substring` 不一样的是，第二个参数是截取多少个
- 从索引 1 开始，截取 3 个，所以得到的是 `ell`

toLowerCase 和 toUpperCase

- 这两个方法分别使用用来给字符串转成 小写字母 和 大写字母 的

```
var str = hello

// 使用 toUpperCase 转换成大写
var upper = str.toUpperCase()

console.log(upper) // HELLO

// 使用 toLowerCase 转换成小写
var lower = upper.toLowerCase()

console.log(lower) // hello
```

强化练习1

1. 实现向数组末尾追加一个元素有多少种方式
2. 理解函数中的 this 指向

强化练习2

1. 给一段文章中的全部指定词语进行过滤

```
// 比如我要过滤 "SM"

var str = 'asdasdSMasdasdasdSMsdasdasdSMsadasd'

```

```
// 需要结果
// asdasd**asdasdasd**sdasdasd**
```

2. 反转字符串

```
var str = 'abcdefg'

// 要求结果
// gfedcba
```

3. 统计字符串中每个字符的个数?

```
var str = 'abcdacbabcbababcbabcabd'

// 结果
// { a: 出现次数， b: 出现次数, ...}
```

强化练习3

4. aabccd统计每个字符出现的次数，去掉重复的字符，使结果显示 abcd

5. 编写函数，判断一个字符串是否是 “可回文字符串”

   - 可回文字符串： 正着和反着一样
   - 例如： `abcba` / `你好世界世好你`
   - 返回值是布尔值