# JavaScript 第六章

## 本章主要内容

> 1. JS中的内置对象
> 2. 数组 Array
> 3. 数学对象 Math
> 4. 时间对象 Date
> 5. 字符串对象 String
> 6. 正则对象 RegExp

## 一、JS中的内置对象

> 在JS中有一个特殊类型的数据是定义好的对象，直接拿来使用

包含有数组、数学对象、时间对象、正则对象

## 二、数组 Array

### 概念

数组是存储数据的仓库，每个储藏室都有一个数字编号(索引 index)

### 定义方式

1. 使用 new Array()定义数组

```
// 如果定义时只传入一个数字，表示声明一个长度为该数字的数组，默认值为undefined
var arr = new Array(10);

// 常规定义
var users = new Array('二郎神','孙悟空','猪八戒');
```

2. 使用[]定义数组

```
// 定义数组
var arr = ['玉皇大帝','王母','宋江'];
```

## 数组长度

```
// 在数组对象中有属性length
var len = arr.length;
```

## 索引 index

```
1. 每一个数组单元的值都是有一个编号对应的，这个编号就称之为索引(index)
2. 索引的值从0开始
3. 索引的最大值为数组长度-1
```

## 数组遍历

1. 使用循环遍历数组

```
for(var i=0;i<arr.length;i++>){
    // i  当前数组的索引
    // arr[i] 就是当前数组的值
}
```

2. 使用forEach()函数遍历数组

```
arr.forEach(function(value,index,arrSelf){
    /*
        第一个参数 value 获取的是当前数组单元的值
        第二个参数 index 获取的是当前数组单元的索引
        第三个参数 arrSelf 获取的是当前数组本身
    */
})
```

## 数组常用方法

1. Array.isArray() 判断是否是数组

```
如果是数组，返回true,
如果不是数组，返回false
```

2. concat() 连接两个或多个数组，返回结果

```
// 连接值
arr.concat('美女');
// 连接数组
arr.concat([2,3],[4,3]);
```

3. join() 将数组单元的值拼接成字符串，可以指定连接符

```
// 默认返回以逗号(,)分割的字符串
arr.join(',')
```

4. reverse() 翻转数组的单元的顺序

```
arr.reverse()
```

5. toString() 将数组转为字符串，并返回结果
6. pop() 弹出数组最后一个单元的值，并返回
7. push() 从数组的尾部压入值，返回新的数组长度
8. shift() 删除并返回数组第一个单元的值
9. unshift() 从数组头部压入值，并返回数组的新长度
10. slice() 从数组中截取
11. splice() 删除数组，并向数组中添加元素
12. indexOf() 正序查询数组单元的值

```
如果存在，返回符合查询条件的第一个结果所在的位置
```

13. lastIndexOf() 倒序查询数组单元的值

```
如果存在，返回符合查询条件的第一个结果所在的位置
如果不存在，返回-1
```

14. map() 映射，原数组单元经回调函数处理后，返回新的结果

```
arr.map(function(value){
    return value*value;
})

// 使用map()获取对象数组中特定的属性值
var users = [
    {
        username : '宋江',
        email : 'songjiang@liangshan.com'
    },
    {
        username : '吴用',
        email : 'zhiduoxing@liangshan.com'
    }
]

// 获取邮箱地址
users.map(function(user){
    return user.email;
})
```

15. filter() 过滤，返回符合条件的结果的值

```
// 获取数组中的偶数
arr.filter(function(i){
    // 返回真，对应的值i，就会返回到新数组中
    return i%2==0;
})
```

16. some() 筛选，只要有一个符合条件就返回true

```
// 只要有一个男的，就返回真
var users = ['男','女','女'];

// 此时res的值为true
var res = users.some(function(value){
    if(value == '男'){
        return true;
    }
});
```

17. every() 筛选，只有当每一个单元都符合条件时才整体返回真

```
// 判断是否全部是男生
var users = ['男','男','女'];

// 此时res的值为false
var res = users.every(function(value){
    if(value == '男'){
        return true;
    }
});
```

18. sort() 数组排序

```
在使用sort()函数时，需要传入对应的比较函数，作为判断条件
```

19. reduce()

```
arr.reduce(function(prevValue,currentValue,currentIndex,arrSelf){

}[,initValue])
```

20. reduceRight()

```
// 等比于reduce()，只不过是从右侧开始计算
```

## 实战案例

1. 计算数组单元的个数
2. 翻转数组
3. 查询出如下格式数组中所有money>1000的人

```
// 数组格式
var uses = [
    {
        username : '张三',
        money : 10000
    }
    ...
];
```

4. 求数组中的最大值
5. 自定义数组排序函数--冒泡排序
6. 数组去重

# 三、数学对象 Math

## 数学常量

```
Math.PI          圆周率 PI
```

## 数学方法

```
Math.random()    获取[0,1)随机数
Math.round()     四舍五入
Math.floor()     舍去取整、向下取整
Math.ceil()      向上取整、进一取整
Math.abs()       返回数值的绝对值
Math.max()       返回所传参数的最大值
Math.min()       返回所传参数的最小值
Math.pow(x,y)    返回x的y次幂
Math.sin()       正弦值
Math.cos()       余弦值
Math.tan()       正切值
```

## 实战案例

1. 定义获取随机数的函数
2. 随机点名器

# 四、时间对象 Date

> 用于处理时间和日期的对象，默认自动将当前的日期时间（时间戳）存储为初始值，该值是从1970年1月1日0:0:0到现在的毫秒数

## 获取时间戳对象

```
// 获取当前此刻的时间戳
var now = new Date();

// 获取指定时间的时间戳

// 1. 传入合法的时间字符串
var time = new Date('2017-1-1 11:0:0');
var time = new Date('2017/2/14 10:00:00');

// 2. 传入时间戳(毫秒数)或者是时间戳对象
var time = new Date(0);
```

### 时间对象常用方法

```
getTime()           // 返回1970年1月1日到指定日期的时间戳，毫秒数
now()               // 返回1970年1月1日到此刻的时间戳
getFullYear()       // 获取年份
getMonth()          // 获取月份(实际月份=getMonth()+1)
getDate()           // 获取日
getHours()          // 获取小时
getMinutes()        // 获取分钟
getSeconds()        // 获取秒
getMilliseconds()   // 获取毫秒
toString()          // 将时间对象转为字符串格式显示
toLocaleString()    // 将时间对象转为本地化的字符串格式显示
```

### 实战案例

1. 网站时间显示
2. 抢购倒计时

# 五、字符串对象 String

### 定义字符串的方式

1. 单引号

   ```
   var str = '床前明月光';
   ```

2. 双引号

   ```
   var str = "疑是地上霜";
   ```

3. 反引号

```
var str = `举头望明月`;
```

4. new String()

```
var str = new String('低头思故乡');
```

> 该方式获取的字符串，使用typeof打印类型为object

## 字符串连接变量

在JS中定义的字符串是不解析变量的，所以使用字符串连接运算符 + 拼接变量与字符串

## 字符串长度

```
var str = 'mumuda';
console.log(str.length);    // 6
```

## 字符串取值

```
// 类似于数组的取值方式
str[index]
```

## 字符串常用方法

```
charAt()                  // 返回指定索引位置的字符
charCodeAt()              // 返回指定字符的unicode值
concat()                  // 拼接两个或多个字符串
indexOf()                 // 正序查找
    如果存在，返回第一个符合条件的数据的索引
    如果不存在，返回-1
lastIndexOf()             // 倒序查找
    如果存在，返回第一个符合条件的数据的索引
    如果不存在，返回-1
trim()                    // 去除首尾空白字符
toUpperCase()             // 全部转为大写
toLowerCase()             // 全部转为小写
slice()                   // 截取字符串
split()                   // 使用指定的分割符分割字符串
match()                   // 使用正则表达式进行匹配，并返回包含结果的数组
repeat()                  // 字符串重复拼接
replace()                 // 替换
search()                  // 查询
substr()                  // 截取
substring()               // 截取
```

### 实战案例

1. 测一测你是王者荣耀的谁

# 六、正则对象 RegExp

### 概念和作用

> 概念：正则表达式是一种描述文本的特定模式(使用一些特定的符号)
>
> 作用：执行字符串的查找匹配以及替换等操作

### 定义正则表达式的方式

1. 使用new RegExp()构造函数

```
var pattern = new RegExp(表达式,模式);
```

2. 使用字面量//定义正则表达式

```
var pattern = /表达式/模式;
```

### 正则表达式的组成部分

> 正则表达式由原子、元字符和模式修正符共同组成

```
原子：描述字符
元字符：修饰描述原子的数量、位置等信息
模式修正符：描述是否要进行全局匹配、多行匹配、是否忽略大小写等
```

## 原子

1. 普通的字符

   ```
   a-z A-Z 0-9 -_等
   ```

2. 非打印字符

   ```
   \n      换行符
   \r      回车符
   \f      换页符
   \s      匹配任何空白字符
   \S      匹配任何非空白符
   \t      制表符
   \w      匹配字母、下划线、数字
   \W      匹配除了 字母、下划线、数字以外的字符
   \d      匹配0-9的数字
   \D      匹配除了0-9数字以外的任意一个字符
   ```

3. 特殊的转义转义字符(转义元字符)

   ```
   转义具有特殊意义的 . * ? ^ $等符号
   ```

## 元字符

> 用于描述原子所在位置、数量、模式单元等信息的符号

描述数量

```
*       匹配0个、1个或多个
?       匹配0个或1个
+       匹配1个或多个
{n}     n为非负整数，匹配恰好n次
{n,}    n为非负整数，匹配至少n次
{m,n}   m,n为非负整数，匹配出现至少m次，最多n次
```

描述位置

```
^       表示开始位置
$       表示结束位置
\b      表示单词边界位置
\B      表示非单词边界位置
```

特殊元字符

```
.       匹配换行符\n以外的任意一个字符
[]      原子列表，匹配列表中的任意一个
[^]     非原子列表，匹配除了列表中字符的任意一个
|       或者
\un     n是4位由16进制数字，用于表示一个unicode字符
```

模式单元

```
()      匹配模式单元，小括号内匹配的内容会进行存储
(?:)    匹配模式单元，小括号匹配的内容不会进行存储
\n      表示模式单元的引用
```

## 模式修正符

```
g       全局匹配
i       忽略大小写
m       表示多行匹配
```

## 贪婪模式和非贪婪模式

贪婪模式:

```
+*元字符在进行模式匹配时，尽可能获取最多的可能
```

非贪婪模式

```
使用特定的方式取消贪婪模式
.*?         取消*的贪婪模式
.+?         取消+的贪婪模式
```

## 正则表达式方法

正则方法

```
test()      检测是否能匹配结果
    存在，返回true
    不存在，返回false

exec()      执行一次正则表达式匹配
    存在，返回包含结果信息的数组
    不存在，返回null
```

字符串方法

```
search()
match()
split()
replace()
```

## 常用正则表达式

1. 邮箱

```
/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
/^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/
```

2. Unicode编码中的汉字范围

```
\u4E00-\u9FA5
```

3. IP地址

```
/((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)/
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]
?[0-9][0-9]?)$/
```

4. HTML标签

```
/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
```

## 实战案例

1. 用户注册信息验证
2. 字符串表情替换为表情图片
3. 数据查询获取(图片地址、文章内容等)