

## 一、流程控制

1. 作用
2. 分类
  - 1) 顺序结构
  - 2) 分支/选择结构
    1. if语句
    2. switch语句
  - 3) 循环结构

## 二、函数

1. 作用
2. 语法
3. 使用
4. 匿名函数
5. 作用域
6. 获取多个DOM元素和控制属性

# 一、流程控制

## 1. 作用

控制代码的执行顺序

## 2. 分类

### 1) 顺序结构

从上到下依次执行代码语句

### 2) 分支/选择结构

#### 1. if语句

- 简单if结构

```
if(条件表达式){  
    表达式成立时执行的代码段  
}
```

注意：除零值以外，其他值都为真，以下条件为假值false

```
if(0){}  
if(0.0){}  
if(""){} //空字符串  
if(undefined){}  
if(NaN){}  
if(null){}
```

特殊写法：{}可以省略，一旦省略，if语句只控制其后的第一行代码

- if - else结构

```
if(条件表达式){
    //条件成立时执行
}else{
    //条件不成立时选择执行
}
```

- 多重分支结构

```
if(条件1){
    //条件1成立时执行
}else if(条件2){
    //条件2成立时执行
}else if(条件3){
    //条件3成立时执行
}...else{
    //条件不成立时执行
}
```

	if 语句	if....else语句	if....else if...else 语句
	只有当指定条件为 true 时，该语句才会执行代码。	在条件为 true 时执行相应代码，否则执行else代码。	根据条件是否成立，执行多个代码块中的相应代码
if...else	<pre>if (条件){     代码块 }</pre>	<pre>if (条件){     代码块 } else{     代码块 }</pre>	<pre>if (条件 1) {     代码块 } else if (条件 2){     代码块 } else{     代码块 }</pre>

2. switch语句

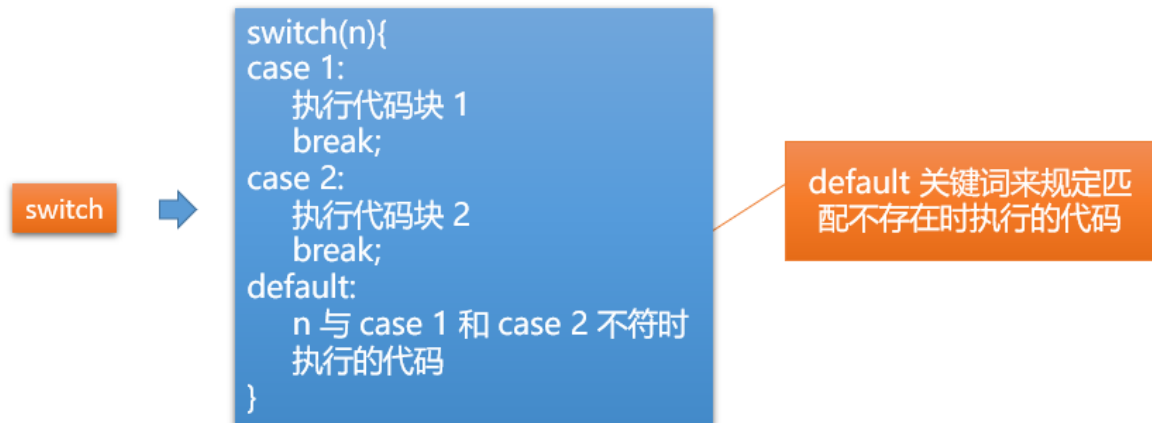
- 语法：

```
switch(value){
    case 值1 :
        //value与值1匹配全等时,执行的代码段
        break; //结束匹配
    case 值2 :
        //value与值2匹配全等时,执行的代码段
        break;
    case 值3 :
        //value与值3匹配全等时,执行的代码段
        break;
    default:
        //所有case匹配失败后默认执行的语句
        break;
}
```

- 使用：

1. **switch**语句用于值的匹配，**case**用于列出所有可能的值；只有**switch()**表达式的值与**case**的值匹配全等时，才会执行**case**对应的代码段
2. **break**用于结束匹配，不再向后执行；可以省略，**break**一旦省略，会从当前匹配到的**case**开始，向后执行所有的代码语句，直至结束或碰到**break**跳出
3. **default**用来表示所有**case**都匹配失败的情况，一般写在末尾，做默认操作
4. 多个**case**共用代码段

```
case 值1:
case 值2:
case 值3:
//以上任意一个值匹配全等都会执行的代码段
```



### 3) 循环结构

- 作用 根据条件，重复执行某段代码
- 分类

#### 1. while循环

```
定义循环变量;
while(循环条件){
    条件满足时执行的代码段
    更新循环变量;
}
```

#### 2. do-while循环

```
do{
    循环体;
    更新循环变量
}while(循环条件);
```

	while 循环	do...while循环
	代码块在指定条件为真时循环执行	该语句在条件为真前已执行一次再检测条件，进行循环执行。
while	<pre>while (条件) {     需要执行的代码 }</pre>	<pre>do{     需要执行的代码 }while (条件);</pre>
	必须确保增加条件中所用变量的值，否则该循环永远不会结束。该可能导致浏览器崩溃。	while与do...while的区别在于后者不管条件是否为真，都将执行一次。

与 while 循环的区别：

- while 循环先判断循环条件，条件成立才执行循环体
- do-while 循环不管条件是否成立，先执行一次循环体



### 3. for 循环

```
for(定义循环变量;循环条件;更新循环变量){  
    循环体;  
}
```

	for 循环	for...in循环
	循环代码块执行指定的次数	该语句循环遍历对象的属性
for	<pre>for (语句 1; 语句 2; 语句 3){     循环的代码块 }</pre>	<pre>for (var in 对象){     循环的代码块 }</pre>
	<p>语句 1 在循环开始前执行，初始化循环中所用变量，是可选项同时，还可初始化任意多个变量。</p> <p>语句 2 定义运行循环的条件，该语句返回 true，则循环再次开始，如果返回 false，则循环将结束，省略是必须用break。</p> <p>语句 3 在循环（代码块）已被执行之后执行，会增加初始变量的值，省略时，代码块中必须有相应的累加值。</p>	<p>for...in 循环中的代码块将针对每个属性执行一次。</p> <p>var 是声明一个变量的var语句，数组的一个元素或者是对象的一个属性名在循环体内部，会被作为字符串赋给变量var。已继承的用户自定义的属性也可以列出。</p>

循环控制：

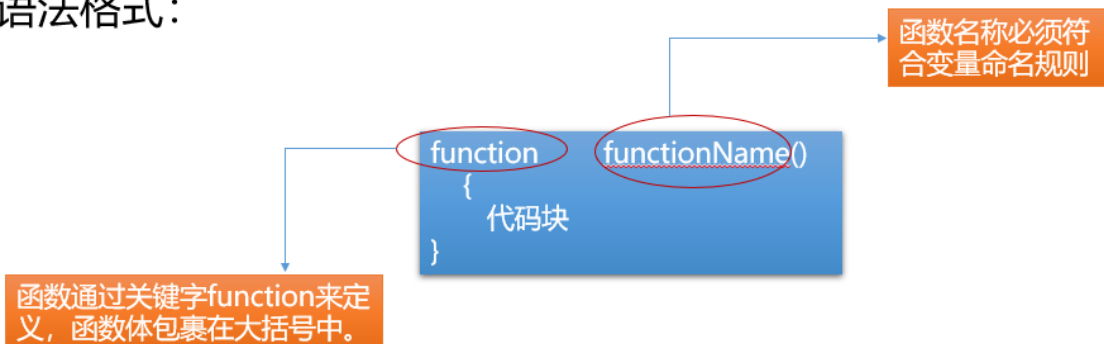
1. break 强制结束循环
2. continue 结束当次循环，开始下一次循环 循环嵌套：在循环中嵌套添加其他循环

break	continue
语句可用于跳出循环，跳出循环后，会继续执行该循环之后的代码，用于for、while、switch	语句中断循环中的遍历，当满足条件条件后继续进行下一次遍历。。
<pre>(条件){     循环体     break; }</pre> 	<pre>(条件){     循环体     (条件)     continue; }</pre> 

总结：continue结束本次循环，循环变量继续递增或递减开始下次循环；而break结束循环，直接执行循环后面的代码。

## 二、函数

- 语法规式：



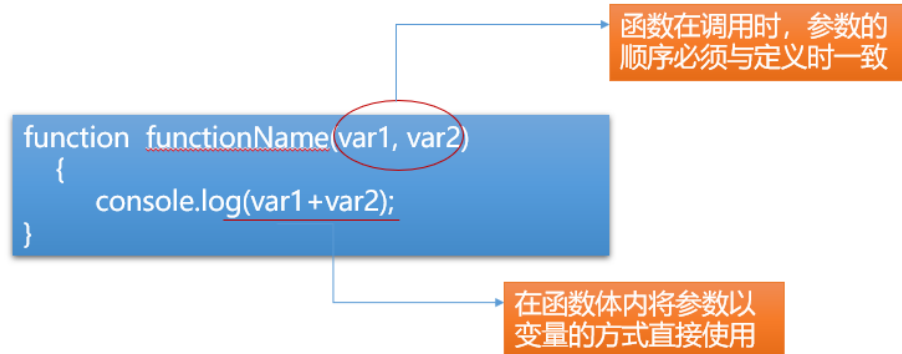
### 1. 作用

封装一段待执行的代码

## 2. 语法

```
//函数声明
function 函数名(参数列表){
    函数体
    return 返回值;
}
//函数调用
函数名(参数列表);
```

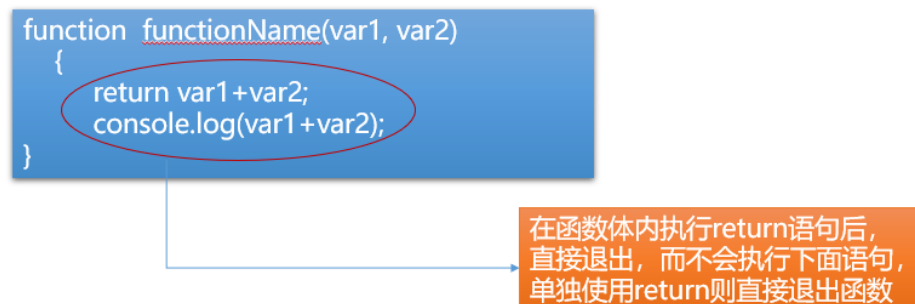
### • 语法规则：



## 3. 使用

函数名自定义，见名知意，命名规范参照变量的命名规范。普通函数以小写字母开头，用于区分构造函数(构造函数使用大写字母开头，定义类)

### • 语法规则：



## 4. 匿名函数

匿名函数：省略函数名的函数。语法为：

- 匿名函数自执行

```
(function (形参){
    //...
})(实参);
```

- 定义变量接收匿名函数

```
var fn = function (){};
fn(); //函数调用
```

• 格式如下：

```
//定义一个匿名函数
(function () {
    //代码块
})
```

执行代码如下：

```
//定义一个匿名函数
(function () {
    //代码块
})()
```

• 匿名函数在执行时，也可以通过括号向函数体传递实参。

## 5. 作用域

JavaScript 中作用域分为全局作用域和函数作用域，以函数的{ }作为划分作用域的依据

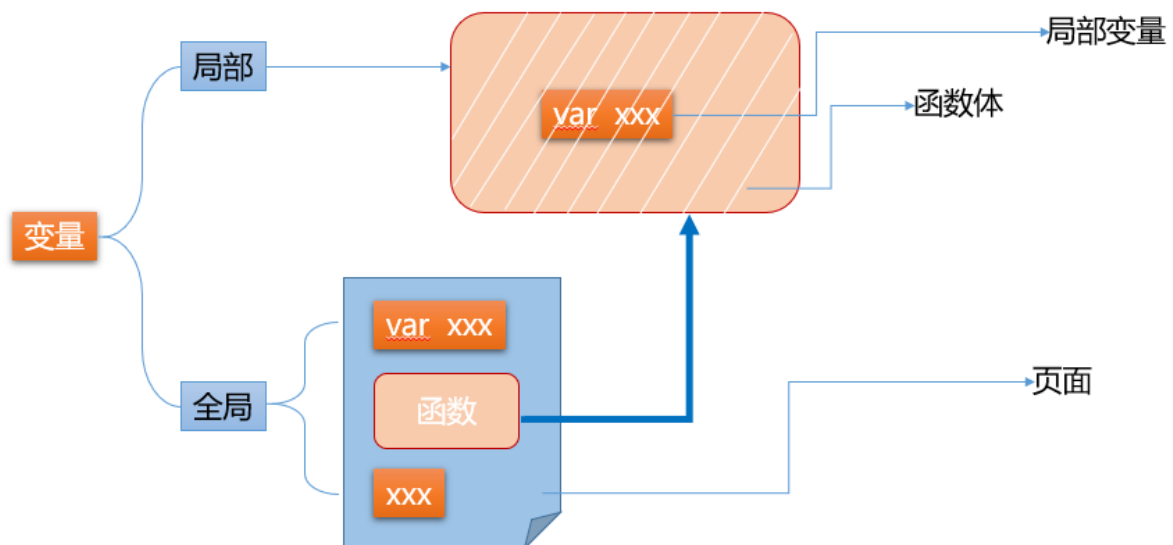
### 1. 全局变量和全局函数

- 只要在函数外部使用 var 关键字定义的变量，或函数都是全局变量和全局函数，在任何地方都可以访问
- 所有省略 var 关键字定义的变量，一律是全局变量

### 2. 局部变量/局部函数

- 在函数内部使用 var 关键字定义的变量为局部变量，函数内部定义的函数也为局部函数，只能在当前作用域中使用，外界无法访问

### 3. 作用域链 局部作用域中访问变量或函数，首先从当前作用域中查找，当前作用域中没有的话，向上级作用域中查找，直至全局作用域



## 6. 获取多个DOM元素和控制属性

### 1. 根据标签名获取元素节点列表

```
var elems = document.getElementsByTagName("");
/*
参数：标签名
返回值：节点列表，需要从节点列表中获取具体的元素节点对象，添加相应下标。
*/
```

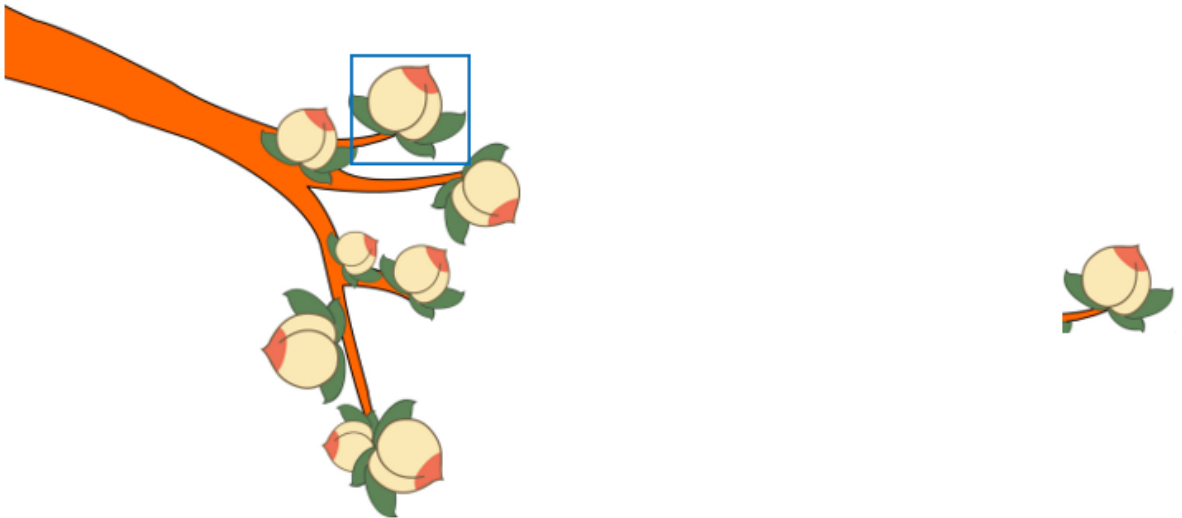
### 2. 根据 class 属性值获取元素节点列表

```
var elems = document.getElementsByClassName("");  
/*  
参数 : 类名(class属性值)  
返回值 : 节点列表  
*/
```

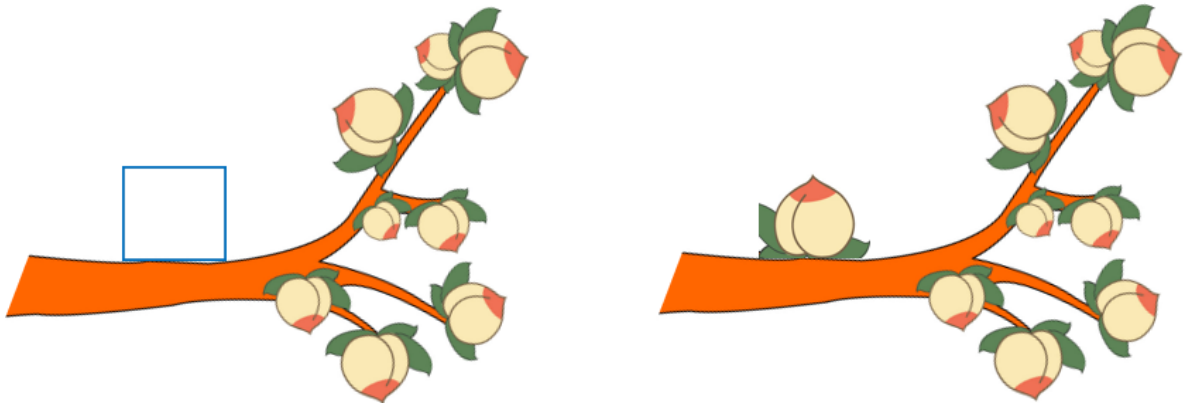
### 3. 元素节点对象提供了以下属性来操作元素内容

**innerHTML** : 读取或设置元素文本内容,可识别标签语法  
**innerText** : 设置元素文本内容,不能识别标签语法  
**value** : 读取或设置表单控件的值

### 4. 获取 DOM 树中的属性值



### 6. 设置 DOM 树中的属性值:



```
elem.getAttribute("attrname");//根据指定的属性名返回对应属性值  
elem.setAttribute("attrname","value");//为元素添加属性,参数为属性名和属性值  
elem.removeAttribute("attrname");//移除指定属性
```