

授课老师：石博文

联系方式：shibw@tedu.cn

授课阶段：Web前端基础

01 javascript

一、JavaScript 概述

1. 什么是JavaScript

- 1) JS 介绍
- 2) JS 组成

2. 使用方式

二、基础语法

1. 语法规则

2. JS的变量与常量

- 1) 变量
- 2) 常量

3. 数据类型

- 1) 基本数据类型（简单数据类型）

4. 数据类型转换

- 1) 强制类型转换
- 2) 隐式类型转换（自动转换）

5. 运算符

- 1) 赋值运算符
- 2) 算数运算符
- 3) 复合运算符
- 4) 自增或自减运算符
- 5) 关系运算符/比较运算符
- 6) 逻辑运算符
- 7) 三目运算符

运算

02

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

匿名函数

03

一、内置对象

1) 对象

2) Array 数组

1. 创建
2. 特点
3. 属性和方法

- 4. 二维数组
- 3) String 对象
 - 1. 创建
 - 2. 特点
 - 3. 属性
 - 4. 方法
- 4) Math 对象
 - 1. 定义
 - 2. 属性
 - 3. 方法
- 5) 日期对象
 - 1. 创建日期对象
 - 2. 日期对象方法

04

- 一、BOM 对象
 - 1. BOM 介绍
 - 2. 对象方法
 - 3. 对象属性
- 二、DOM节点操作
 - 1. 节点对象
 - 2. 访问节点
 - 3. 操作元素样式

05

- jQuery简介
 - 1. 介绍
 - 2. 使用
 - 1) 引入
 - 2) 工厂函数 - \$()
 - 3) 原生JavaScript对象与jQuery对象
 - 4) jQuery获取元素
 - 5) 操作元素内容
 - 6) 操作标签属性
 - 7) 操作标签样式
 - 8) 元素的创建,添加,删除
 - 9) 动画效果
 - 10) 数据与对象遍历
 - 11) jQuery事件处理
- 3.实战
 - 1. 页面效果
 - 2. 代码分析
 - 1. 页面元素
 - 2. 初始代码
 - 3. 绑定省份
 - 4. 绑定城市

01 javascript

一、JavaScript 概述

1. 什么是JavaScript

1) JS 介绍

简称JS，是一种浏览器解释型语言,嵌套在HTML文件中交给浏览器解释执行。主要用来实现网页的动态效果，用户交互及前后端的数据传输等。

2) JS 组成

1. 核心语法 -ECMAScript 规范了JS的基本语法
2. 浏览器对象模型 -BOM
Browser Object Model，提供了一系列操作浏览器的方法
3. 文档对象模型 -DOM
Document Object Model，提供了一系列操作的文档的方法

2. 使用方式

1. 元素绑定事件

- 事件：指用户的行为（单击，双击等）或元素的状态（输入框的焦点状态等）
- 事件处理：元素监听某种事件并在事件发生后自动执行事件处理函数。
- 常用事件：onclick (单击事件)
- 语法：将事件名称以标签属性的方式绑定到元素上，自定义事件处理。

```
1 <!--实现点击按钮在控制台输出-->
2 <button onclick="console.log('Hello world');">点击</button>
```

2. 文档内嵌。使用标签书写 JS 代码

- 语法：

```
1 <script type="text/javascript">
2   alert("网页警告框");
3 </script>
```

- 注意：标签可以书写在文档的任意位置，书写多次，一旦加载到script标签就会立即执行内部的JS代码，因此不同的位置会影响代码最终的执行效果

3. 外部链接

- 创建外部的JS文件 XX.js，在HTML文档中使用引入

```
1 <script src="index.js"></script>
```

- 注意：既可以实现内嵌JS代码，也可以实现引入外部的JS文件，但是只能二选一。

二、基础语法

1. 语法规范

1. JS是由语句组成,语句由关键字,变量,常量,运算符,方法组成.分号可以作为语句结束的标志,也可以省略
2. JS严格区分大小写
3. 注释语法
单行注释使用 //
多行注释使用 /* */

2. JS的变量与常量

1) 变量

1. 作用：用于存储程序运行过程中可动态修改的数据
2. 语法：使用关键var声明,自定义变量名

```
1 var a;           //变量声明
2 a = 100;         //变量赋值
3 var b = 200;     //声明并赋值
4 var m,n,k;       //同时声明多个变量
5 var j = 10,c = 20; //同时声明并赋值多个变量
```

3. 命名规范：

- 变量名,常量名,函数名,方法名自定义,可以由数字,字母,下划线,\$组成,禁止以数字开头
- 禁止与关键字冲突(var const function if else for while do break case switch return class)
- 变量名严格区分大小写
- 变量名尽量见名知意,多个单词组成采用小驼峰,例如: "userName"

4. 使用注意：

- 变量如果省略var关键字,并且未赋值,直接访问会报错
- 变量使用var关键字声明但未赋值,变量初始值为undefined
- 变量省略var关键字声明,已被赋值,可正常使用.影响变量作用域

2) 常量

1. 作用：存储一经定义就无法修改的数据
2. 语法：必须声明的同时赋值

```
1 const PI = 3.14;
```

3. 注意：

- 常量一经定义,不能修改,强制修改会报错
- 命名规范同变量,为了区分变量,常量名采用全大写字母

3. 数据类型

1) 基本数据类型（简单数据类型）

1. number 数值类型

- 整数

1. 十进制表示

```
1 var a = 100;
```

```
1 2. 八进制表示
2 以0为前缀
```

```
1 var b = 021; //结果为十进制的 17
```

```
1 3. 十六进制
2 以0x为前缀
```

```
1 | var c = 0x35; //结果为十进制的 53
```

```
1 | 使用 : 整数可以采用不同进制表示,在控制台输出时一律会按照十进制输出
```

○ 小数

1. 小数点表示

```
1 | var m = 1.2345;
```

2. 科学计数法

例: 1.5e3

e表示10为底,e后面的数值表示10的次方数

1.5e3 等价于 $1.5 * 10(3)$

2. string 字符串类型

字符串: 由一个或多个字符组成,使用""或"表示,每一位字符都有对应的Unicode编码

```
1 | var s = "100";  
2 | var s1 = "张三";
```

3. boolean 布尔类型

只有真和假两个值,布尔值与number值可以互相转换。true 为 1, false 为 0

```
1 | var isSave = true;  
2 | var isChecked = false;
```

4. undefined (程序返回的值)

特殊值,变量声明未赋值时显示undefined

```
1 | var a;  
2 | console.log(a); //undefined
```

```
1 |  
2 | 5. null 空类型 (主动使用的)  
3 | 解除对象引用时使用null,表示对象为空  
4 | ##### 2) 引用数据类型  
5 | 主要指对象,函数等  
6 | ##### 3) 检测数据类型  
7 | typeof 变量或表达式  
8 | typeof (变量或表达式)  
9 |  
10 |  
11 | ```javascript  
12 | var n = "asda";  
13 | console.log(typeof n); //string  
14 | console.log(typeof(n)); //string
```

4. 数据类型转换

不同类型的数据参与运算时,需要转换类型

1) 强制类型转换

1. 转换字符串类型

方法: toString()

返回转换后的字符串

```
1 var a = 100;
2 a = a.toString(); //"100"
3 var b = true;
4 b = b.toString(); //"true"
```

2. 转换number类型

o Number(param)

参数为要进行数据类型转换的变量或值, 返回转换后的结果:

如果转换成功,返回number值

如果转换失败,返回NaN,(Not a Number), 只要数据中存在非number字符,一律转换失败, 返回 NaN

```
1 typeof NaN
2 "number"
3 Number(undefined)
4 NaN
5 Number(null)
6 0
7 Number(true)
8 1
9 Number(false)
10 0
11 Number()
12 0
13 Number('')
14 0
15 Number('  ')
16 0
17 Number("abc")
18 NaN
```

o parseInt(param)

参数为要解析的数据

作用: 从数据中解析整数值

过程:

1. 如果参数为非字符串类型,会自动转成字符串
- 2 左向右依次对每一位字符转number, 转换失败则停止向后解析, 返回结果

```
1 parseInt()
2 NaN
3 parseInt(0.6)
4 0
5 parseInt(1.8)
6 1
7 parseInt('asd')
8 NaN
9 parseInt('12qw')
```

```

10 12
11 parseInt('12qw12')
12 12
13 parseInt(true)
14 NaN
15 parseInt(false)
16 NaN
17 parseInt(undefined)
18 NaN
19 parseInt(null)
20 NaN
21 parseInt('')
22 NaN

```

- parseFloat(param)
作用：提取number值，包含整数和小数部分

1 | 与parseInt()类似

2) 隐式类型转换（自动转换）

1. 当字符串与其他数据类型进行"+"运算时,表示字符串的拼接，不再是数学运算
转换规则：将非字符串类型的数据转换成字符串之后进行拼接，最终结果为字符串
2. 其他情况下，一律将操作数转number进行数学运算

5. 运算符

1) 赋值运算符

1 | = 将右边的值赋给左边变量

2) 算数运算符

1 | + - * / % 加 减 乘 除 取余

3) 复合运算符

1 | += -= *= /= %=

4) 自增或自减运算符

1 | ++ -- 变量的自增和自减指的是在自身基础上进行 +1或-1 的操作

注意：

- 自增或自减运算符在单独与变量结合时，放前和放后没有区别
- 如果自增或自减运算符与其他运算符结合使用，要区分前缀和后缀,做前缀，那就先++/--,再进行赋值或其他运算，如果做后缀，就先结合其他运算符，再进行++ / --

5) 关系运算符/比较运算符

```
1 > <
2 >= <=
3 ==(相等) !=(不相等)
4 ===(全等) !==(不全等)
```

1. 关系运算符用来判断表达式之间的关系,结果永远是布尔值 true/false

2. 使用

- 字符串与字符串之间的比较
依次比较每位字符的Unicode码,只要某位字符比较出结果,就返回最终结果
- 其他情况
一律将操作数转换为number进行数值比较, 如果某一操作数无法转换number, 则变成NaN
参与比较运算, 结果永远是false

null和其他数据类型做等值比较运算 不转换成数字

null和undefined相等 但是 null和undefined不全等

3. 相等与全等

- 相等: 不考虑数据类型,只做值的比较(包含自动类型转换)
- 全等: 不会进行数据类型转换,要求数据类型一致并且值相等才判断全等

6) 逻辑运算符

1. && 逻辑与 条件1&&条件2 (and)

表达式同时成立,最终结果才为true;全1则1

2. || 逻辑或 条件1||条件2 (or)

表达式中只要有一个成立,最终结果即为true; 有1则1

3. ! 逻辑非 !条件 (not)

对已有表达式的结果取反

注意: 除零值以外,所有值都为真

7) 三目运算符

语法:

```
1 表达式1 ? 表达式2 : 表达式3;
```

过程:

判断表达式1是否成立,返回布尔值

如果表达式1成立,执行表达式2;

如果表达式1不成立,执行表达式3;

a>b?console.log('a比b大');console.log('b比a大')

```
1      2      3
```

运算

```
1 true + false
2 1
3 undefined == null;
4 true
```



```
5 | undefined === null;
6 | false
7 | null < 1;
8 | true
9 | null == 0;
10 | false
11 | NaN == NaN (NaN 与谁比较都是false)
12 | false
13 | NaN ===NaN
14 | false
```

02

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

```
1 | if(条件表达式){
2 |     表达式成立时执行的代码段
3 | }
```

注意：除零值以外，其他值都为真，以下条件为假值false

```
1 | if(0){}
2 | if(0.0){}
3 | if(""){} //空字符串
4 | if(undefined){}
5 | if(NaN){}
6 | 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

```

1  if(条件表达式){
2      //条件成立时执行
3  }else{
4      //条件不成立时选择执行
5  }

```

- 多重分支结构

```

1  if(条件1){
2      //条件1成立时执行
3  }else if(条件2){
4      //条件2成立时执行
5  }else if(条件3){
6      //条件3成立时执行
7  }...else{
8      //条件不成立时执行
9  }

```

	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：

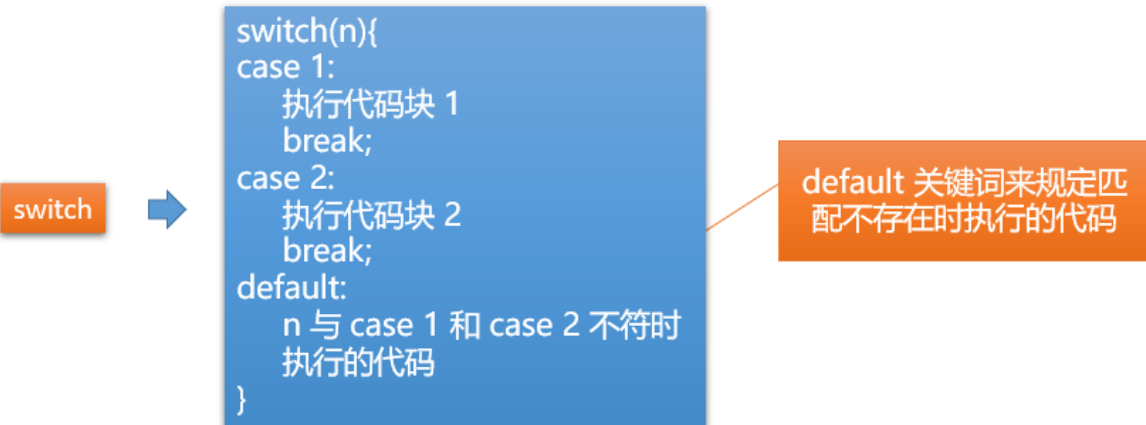
```

1  switch(value){
2      case 值1 :
3      //value与值1匹配全等时,执行的代码段
4      break; //结束匹配
5      case 值2 :
6      //value与值2匹配全等时,执行的代码段
7      break;
8      case 值3 :
9      //value与值3匹配全等时,执行的代码段
10     break;
11     default:
12     //所有case匹配失败后默认执行的语句
13     break;
14 }

```

- 使用：

- 1 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 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 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 4. 多个**case**共用代码段
- ```
5 case 值1:
6 case 值2:
7 case 值3:
8 //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

```
1 定义循环变量;
2 while(循环条件){
3 条件满足时执行的代码段
4 更新循环变量;
5 }
```

#### 2. do-while循环

```
1 do{
2 循环体;
3 更新循环变量
4 }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

```
1 for(定义循环变量;循环条件;更新循环变量){
2 循环体;
3 }
```



|     | 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

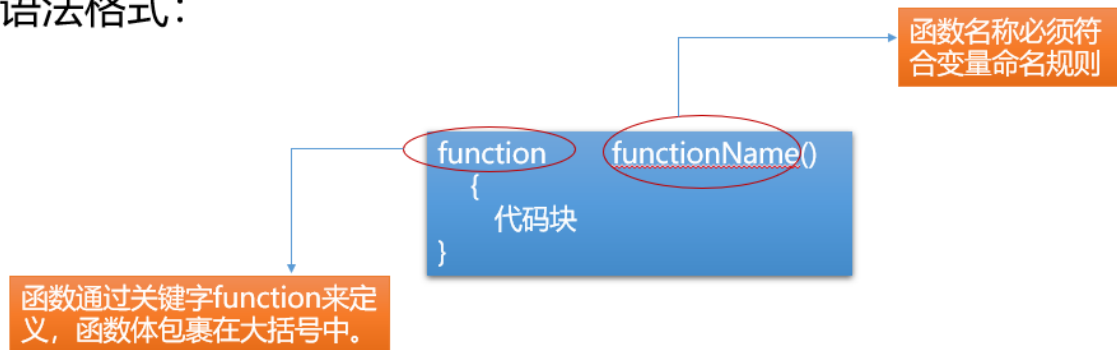
循环嵌套：

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>  |
| <p>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。</p>                                                         |                                                                                                                                          |

## 二、函数

## • 语法规则：



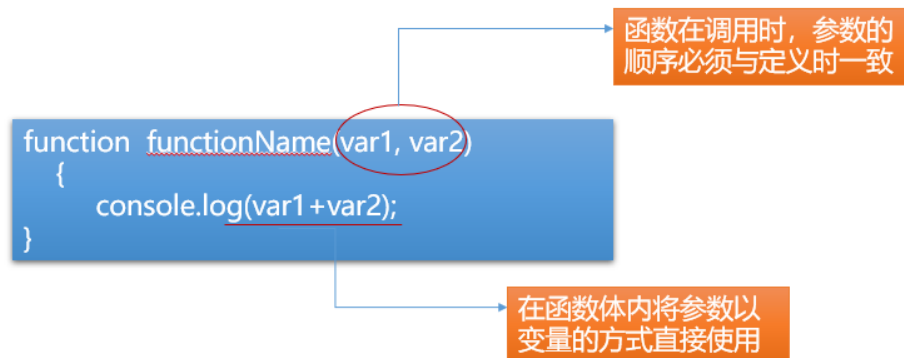
## 1. 作用

封装一段待执行的代码

## 2. 语法

```
1 //函数声明
2 function 函数名(参数列表){
3 函数体
4 return 返回值;
5 }
6 //函数调用
7 函数名(参数列表);
```

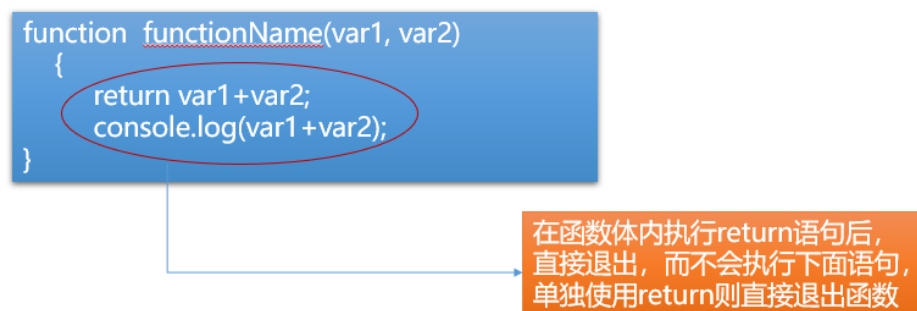
## • 语法规则：



## 3. 使用

- 1 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

```
1 (function (形参){
2
3 })(实参);
```

- 定义变量接收匿名函数

```
1 var fn = function (){};
2 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

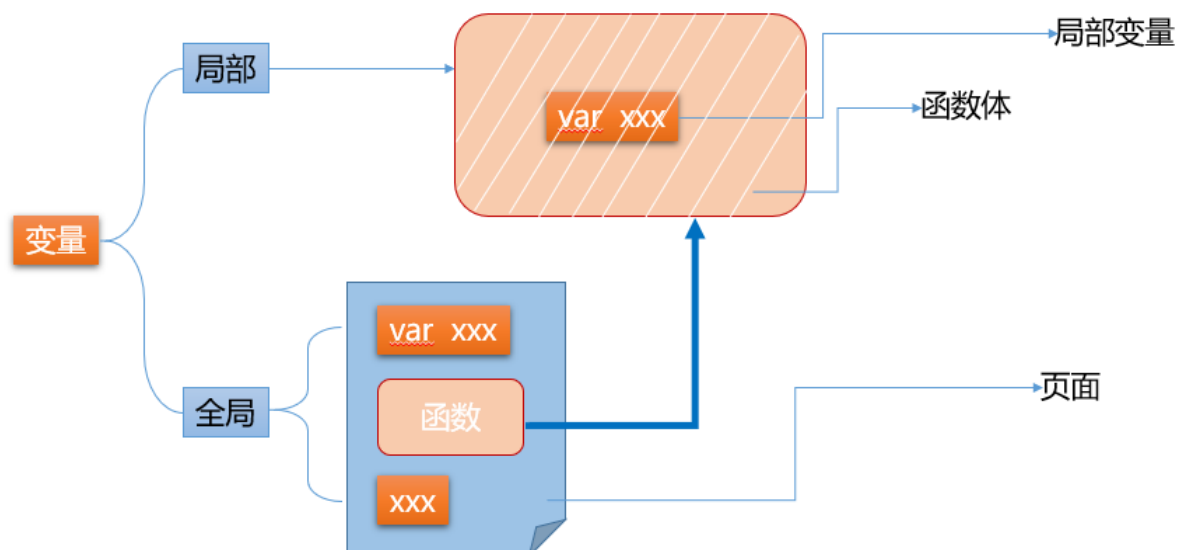
- 只要在函数外部使用 var 关键字定义的变量，或函数都是全局变量和全局函数，在任何地方都可以访问
- 所有省略 var 关键字定义的变量，一律是全局变量

### 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

```
1 var elems = document.getElementsByTagName("");
2 /*
3 参数 : 标签名
4 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。
5 */
```

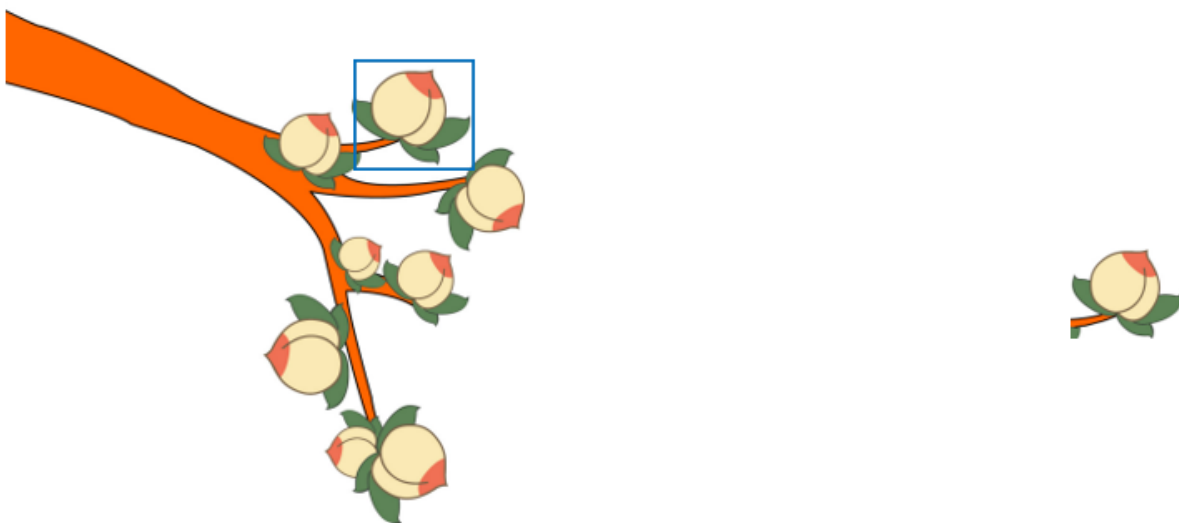
### 2. 根据 class 属性值获取元素节点列表

```
1 var elems = document.getElementsByClassName("");
2 /*
3 参数 : 类名(class属性值)
4 返回值 : 节点列表
5 */
```

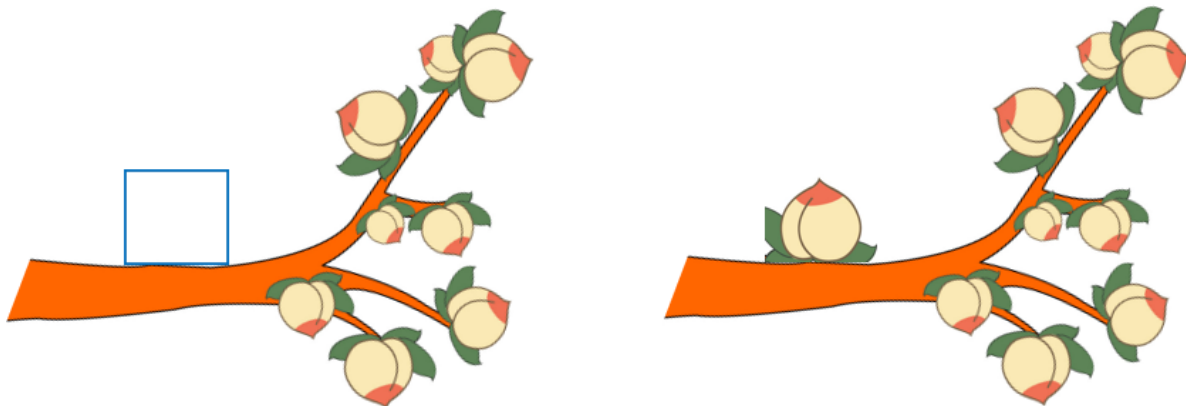
### 3. 元素节点对象提供了以下属性来操作元素内容

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

### 4. 获取 DOM 树中的属性值



### 5. 设置 DOM 树中的属性值:



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

## 匿名函数

```
1 <!-- <button id="btn" onclick="sayHello()">click me</button> -->
2 <button id="btn" onclick="">click me</button> onclick属性
```

## 03

### 一、 内置对象

#### 1) 对象

对象是由属性和方法组成的,使用点语法访问

#### 2) Array 数组

##### 1. 创建

##### 2. 特点

- 数组用于存储若干数据,自动为每位数据分配下标,从0开始
- 数组中的元素不限数据类型,长度可以动态调整
- 动态操作数组元素：根据元素下标读取或修改数组元素，arr[index]

##### 3. 属性和方法

1. 属性：length 表示数组长度,可读可写

2. 方法：

- push(data)  
在数组的末尾添加一个或多个元素,多个元素之间使用逗号隔开  
返回添加之后的数组长度
- pop()  
移除末尾元素  
返回被移除的元素
- unshift(data)  
在数组的头部添加一个或多个元素  
返回添加之后的数组长度
- shift()  
移除数组的第一个元素  
返回被移除的元素
- splice(index,num)  
从数组中添加/删除项目  
返回被删除的项目



- toString()  
将数组转换成字符串类型  
返回字符串结果
- join(param)  
将数组转换成字符串,可以指定元素之间的连接符,如果参数省略,默认按照逗号连接  
返回字符串
- reverse()  
反转数组,倒序重排  
返回重排的数组,注意该方法直接修改原数组的结构
- sort()  
对数组中元素排序,默认按照Unicode编码升序排列  
返回重排后的数组,直接修改原有数组  
参数: 可选,自定义排序算法  
例:

```
1 //自定义升序
2 function sortASC(a,b){
3 return a-b;
4 }
```

作用: 作为参数传递到sort()中,会自动传入两个元素进行比较,如果 $a-b>0$ ,交换元素的值,自定义升序排列

```
1 //自定义降序
2 function sortDESC(a,b){
3 return b-a;
4 }
5 //如果返回值>0,交换元素的值,b-a表示降序排列
```

## 4. 二维数组

数组中的每个元素又是数组

```
1 var arr1 = [1,2,3];
2 var arr2 = [[1,2],[3,4],[5,6,7]];
3 //操作数组元素
4 var r1 = arr2[0] //内层数组
5 var num = r1[0]; //值 1
6 //简写
7 var num2 = arr2[1][0];
```

## 3) String 对象

### 1. 创建

```
1 var str = "100";
2 var str2 = new String("hello");
```

## 2. 特点

字符串采用数组结构存储每位字符,自动为字符分配下标,从0开始

## 3. 属性

length : 获取字符串长度

## 4. 方法

- 转换字母大小写
  - toUpperCase() 转大写字母
  - toLowerCase() 转小写字母
  - 返回转换后的字符串,不影响原始字符串
- 获取字符或字符编码
  - charAt(index) 获取指定下标的字符
  - charCodeAt(index) 获取指定下标的字符编码
  - 参数为指定的下标,可以省略,默认为0
- 获取指定字符的下标
  - indexOf(str,fromIndex)
    - 作用: 获取指定字符的下标,从前向后查询,找到即返回
    - 参数:
      - str 表示要查找的字符串,必填
      - fromIndex 表示起始下标,默认为0
    - 返回:
      - 返回指定字符的下标,查找失败返回-1
- 截取字符串
  - substring(startIndex,endIndex)
    - 作用: 根据指定的下标范围截取字符串,startIndex ~ endIndex-1
    - 参数:
      - startIndex 表示起始下标
      - endIndex 表示结束下标,可以省略,省略表示截止末尾
- substr(startIndex,len)
  - 作用: 根据下标截取指定的字符串
- 分割字符串
  - split(param)
    - 作用: 将字符串按照指定的字符进行分割,以数组形式返回分割结果
    - 参数: 指定分隔符,必须是字符串中存在的字符,如果字符串中不存在,分割失败,仍然返回数组
- 模式匹配
- 正则表达式对象 RegExp
  - RegExp : Regular Expression
  - 1. 语法:
    - var reg1 = /微软/ig;
    - var reg2 = new RegExp('匹配模式','修饰符');
    - 正则表达式对象可以接收一个变量。
  - 2. 属性:
    - lastIndex : 可读可写, 表示下一次匹配的起始索引
    - 注意:

1. 默认情况下，正则表达式对象不能重复调用方法，如果重复调用，结果会出错：  
由于 lastIndex 保存再一次匹配的起始下标，重复调用时，不能保证每次都从下标0开始验证，可以手动调整 lastIndex 为 0。
  2. 只有正则对象设置全局匹配 g，该属性才起作用。
3. 方法：
- test(str) :验证字符串中是否存在满足正则匹配模式的内容，存在则返回true，不存在返回false参数为要验证的字符串。
- 作用：借助正则表达式实现字符串中固定格式内容的查找和替换
- 正则表达式：
- var reg1 = /字符模式/修饰符;
- 修饰符：
- i : ignorecase 忽略大小写
- g : global 全局范围
- 字符串方法：
- match(regExp/subStr)  
作用：查找字符串中满足正则格式或满足指定字符串的内容  
返回：数组,存放查找结果
  - replace(regExp/subStr,newStr)  
作用：根据正则表达式或字符串查找相关内容并进行替换  
返回：替换后的字符串,不影响原始字符串。

## 4) Math 对象

---

### 1. 定义

Math对象主要提供一些列数学运算的方法

### 2. 属性

1. 圆周率：Math.PI
2. 自然对数：Math.E

### 3. 方法

1. Math.random(); 生成0-1之间的随机数
2. Math.ceil(x); 对x向上取整,忽略小数位,整数位+1
3. Math.floor(x); 对x向下取整,舍弃小数位,保留整数位
4. Math.round(x); 对x四舍五入取整数

## 5) 日期对象

---

### 1. 创建日期对象

```
1 1. var date2 = new Date("2011/11/11");
2 2. var date3 = new Date("2011/11/11 11:11:11");
```

## 2. 日期对象方法

1. 读取或设置当前时间的毫秒数: `getTime()`
2. 获取时间分量
  - `getFullYear()`
  - `getMonth()`
  - `getDate()`
  - `getDay()`获取星期

# 04

## 一、BOM 对象

### 1. BOM 介绍

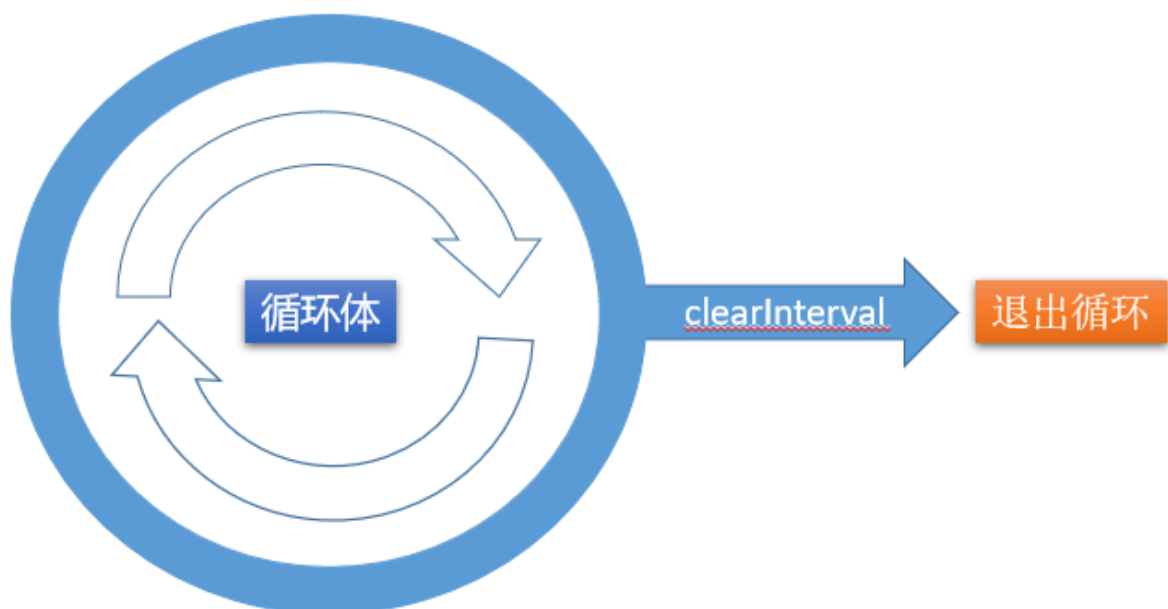
BOM全称为“Browser Object Model”，浏览器对象模型。提供一系列操作浏览器的属性和方法。核心对象为window对象，不需要手动创建，跟随网页运行自动产生，直接使用，在使用时可以省略书写。

### 2. 对象方法

#### 1. 网页弹框

```
1 alert() //警告框
2 confirm() //确认框
```

#### 2. 定时器方法



#### 周期性定时器

作用：每隔一段时间就执行一次代码

```

1 //开启定时器:
2 var timerID = setInterval(function,interval);
3 /*
4 参数 :
5 function : 需要执行的代码,可以传入函数名;或匿名函数
6 interval : 时间间隔,默认以毫秒为单位 1s = 1000ms
7 返回值 : 返回定时器的ID,用于关闭定时器
8 */

```

关闭定时器:

```

1 //关闭指定id对应的定时器
2 clearInterval(timerID);

```



### 一次性定时器

作用: 等待多久之后执行一次代码

```

1 //开启超时调用:
2 var timerId = setTimeout(function,timeout);
3 //关闭超时调用:
4 clearTimeout(timerId);

```

## 3. 对象属性

window的大部分属性又是对象类型

### 1. history

作用: 保存当前窗口所访问过的URL

属性: length 表示当前窗口访问过的URL数量

方法:

```

1 back() 对应浏览器窗口的后退按钮, 访问前一个记录
2 forward() 对应前进按钮, 访问记录中的下一个URL

```

### 2. location

作用: 保存当前窗口的地址栏信息(URL)

属性: href 设置或读取当前窗口的地址栏信息

方法:

```

1 reload(param) 重载页面(刷新)
2 参数为布尔值, 默认为 false, 表示从缓存中加载, 设置为true, 强制从服务器根目录加载

```

## 二、DOM节点操作

DOM全称为“Document Object Model”，文档对象模型，提供操作HTML文档的方法。（注：每个html文件在浏览器中都视为一篇文档,操作文档实际就是操作页面元素。）

### 1. 节点对象

JavaScript 会对 html 文档中的元素、属性、文本甚至注释进行封装，称为节点对象，提供相关的属性和方法。

### 2. 访问节点

- 元素节点（操作标签）
- 属性节点（操作标签属性）
- 文本节点（操作标签的文本内容）

标签属性都是元素节点对象的属性,可以使用点语法访问，例如：

```
1 h1.id = "d1"; //set 方法
2 console.log(h1.id); //get 方法
3 h1.id = null; //remove 方法
```

注意：

- 属性值以字符串表示
- class属性需要更名为 className，避免与关键字冲突，例如：  
h1.className = "c1 c2 c3";

### 3. 操作元素样式

1. 为元素添加 id、class属性，对应选择器样式
2. 操作元素的行内样式，访问元素节点的style属性，获取样式对象；样式对象中包含CSS属性，使用点语法操作。

```
1 p.style.color = "white";
2 p.style.width = "300px";
3 p.style.fontSize = "20px";
```

注意：

- 属性值以字符串形式给出，单位不能省略
- 如果css属性名包含连接符，使用JS访问时，一律去掉连接符,改为驼峰，font-size -> fontSize

## 05

## jQuery简介

### 1. 介绍

jQuery是JavaScript的工具库，对原生JavaScript中的DOM操作、事件处理、包括数据处理和Ajax技术等  
进行封装,提供更完善，更便捷的方法。

### 2. 使用

## 1) 引入

先引入jquery文件，才能使用jquery语法

1. CDN 有网（备用）
2. 本地文件（常用）

## 2) 工厂函数 - \$()

"\$()"函数用于获取元素节点，创建元素节点或将原生JavaScript对象转换为jquery对象,返回 jQuery 对象。jQuery 对象实际是一个类数组对象，包含了一系列 jQuery 操作的方法。

例如：

```
1 //$() 获取元素节点,需传入字符串的选择器
2 $("h1")
3 $("#d1")
4 $(".c1")
5 $("body,h1,p")
6 //选择器的特点，与样式选择器一致
```

## 3) 原生JavaScript对象与jQuery对象

原生JavaScript对象与jQuery对象的属性和方法不能混用。可以根据需要，互相转换：

1. 原生JavaScript转换jQuery对象  
\$(原生对象)，返回 jQuery 对象
2. jQuery对象转换原生JavaScript对象
  - 方法一：根据下标取元素,取出即为原生对象  
var div = \$("div")[0];
  - 方法二：使用jQuery的get(index)取原生对象  
var div2 = \$("div").get(0);

## 4) jQuery获取元素

jQuery通过选择器获取元素，\$("选择器")

选择器分类：

### 1. 基础选择器

```
1 标签选择器: $("div")
2 ID 选择器: $("#d1")
3 类选择器: $(".c1")
4 群组选择器: $("body,p,h1")
```

### 2. 层级选择器

```
1 后代选择器: $("div .c1")
2 子代选择器: $("div>span")
3 相邻兄弟选择器: $("h1+p") 匹配选择器1后的第一个兄弟元素,同时要求兄弟元素满足选择器2
4 通用兄弟选择器: $("h1~h2") 匹配选择器1后所有满足选择器2的兄弟元素
```

### 3. 过滤选择器，需要结合其他选择器使用。

```
1 :first
2 匹配第一个元素 例: $("p:first")
```

```

3 :last
4 匹配最后一个元素 例:$("p:last")
5 :odd
6 匹配奇数下标对应的元素
7 :even
8 匹配偶数下标对应的元素
9 :eq(index)
10 匹配指定下标的元素
11 :lt(index)
12 匹配下标小于index的元素
13 :gt(index)
14 匹配下标大于index的元素
15 :not(选择器)
16 否定筛选,除()中选择器外,其他元素

```

## 5) 操作元素内容

```

1 html() //设置或读取标签内容,等价于原生innerHTML,可识别标签语法
2 text() //设置或读取标签内容,等价于innerText,不能识别标签
3 val() //设置或读取表单元素的值,等价于原生value属性

```

## 6) 操作标签属性

1. attr("attrName","value")  
设置或读取标签属性
2. prop("attrName","value")  
设置或读取标签属性  
注意:在设置或读取元素属性时,attr()和prop()基本没有区别;但是在读取或设置表单元素(按钮)的选中状态时,必须用prop()方法,attr()不会监听按钮选中状态的改变,只看标签属性checked是否书写
3. removeAttr("attrName")  
移除指定属性

## 7) 操作标签样式

1. 为元素添加id/class属性,对应选择器样式
2. 针对类选择器,提供操作class属性值的方法

```

1 addClass("className") //添加指定的类名
2 removeClass("className")//移除指定的类型,如果参数省略,表示清空class属性值
3 toggleClass("className")//结合用户行为,实现动态切换类名.如果当前元素存在指定类名,则移除;不存在则添加

```

### 3. 操作行内样式

```

1 css("属性名","属性值") //设置行内样式
2 css(jQuery对象) //设置一组CSS样式
3 /*
4 JavaScript对象:常用数据传输格式
5 语法:
6 {
7 "width":"200px",
8 "height":"200px",
9 "color":"red"
10 }
11 */

```



## 8) 元素的创建,添加,删除

1. 创建：使用\$("标签语法")，返回创建好的元素

```
1 | var div = $("

</div>"); //创建元素
2 | div.html("动态创建").attr("id","d1").css("color","red"); //链式调用，设置内容和属性
3 | var h1 = $("

#

2. 作为子元素添加


```
1 | $obj.append(newObj); //在$obj的末尾添加子元素newObj
2 | $obj.prepend(newObj); //作为第一个子元素添加至$obj中
```


3. 作为兄弟元素添加


```
1 | $obj.after(newObj); //在$obj的后面添加兄弟元素
2 | $obj.before(newObj); //在$obj的前面添加兄弟元素
```


4. 移除元素


```
1 | $obj.remove(); //移除$obj
```


9) 动画效果

1. 显示和隐藏


```
1 | show(speed,callback)/hide(speed,callback)
```


- speed 可选。规定元素从隐藏到完全可见的速度。默认为 "0"。
- callback 可选。show 函数执行完之后，要执行的函数

2. 通过使用滑动下拉和上推效果，显示隐藏的被选元素（只针对块元素）


```
1 | slideDown(speed,callback)/slideUp(speed,callback)
```


3. 通过使用淡隐淡现方式显示效果，显示隐藏的被选元素


```
1 | fadeOut(speed,callback)/fadeIn(speed,callback)
```


4. 动画函数，可以实现自定义动画 animate 函数


```
1 | animate(styles,speed,callback)
```


- styles 必需。规定产生动画效果的 CSS 样式和值
- speed 可选。规定动画的速度。默认是 "normal"
- callback 可选。show 函数执行完之后，要执行的函数


```

## 10) 数据与对象遍历

1. `$(selector).each()` 方法规定为每个匹配元素规定运行的函数

```
1 | $(selector).each(function(index,element){})
```

必需。为每个匹配元素规定运行的函数。

- `index` - 选择器的 `index` 位置
- `element` - 当前的元素

2. `$.each()` 函数是框架提供的一个工具类函数，通过它，你可以遍历对象、数组的属性值并进行处理

```
1 | $.each(object, function(index, data){});
```

必需。为每个匹配元素规定运行的函数。

- `index` - 选择器的 `index` 位置
- `data` - 当前的数据

## 11) jQuery事件处理

1. 文档加载完毕

原生JavaScript 方法: `window.onload`

jQuery:

```
1 | //语法一
2 | $(document).ready(function (){
3 | //文档加载完毕后执行
4 | })
5 | //语法二
6 | $.ready(function (){
7 | //文档加载完毕后执行
8 | })
9 | //语法三
10 | $(function(){
11 | //文档加载完毕后执行
12 | })
```

区别:

原生`onload`事件不能重复书写，会产生覆盖问题；jquery中对事件做了优化,可以重复书写`ready`方法,依次执行

2. 事件绑定方式

事件名称省略 `on` 前缀

```
1 | //on("事件名称", function)
2 | $("div").on("click",function(){}); //新版本使用的多些
3 | //bind("事件名称",function)
4 | $("div").bind("click",function(){}); //1.6-1.8间的版本
5 | //事件名作为方法名
6 | $("div").click(function(){});
```

3. `this` 表示事件的触发对象，在jquery中可以使用，注意转换类型。`this` 为原生对象只能使用原生的属性和方法，可以使用 `$(this)` 转换为jquery对象，使用jquery方法。

## 3.实战

### 1. 页面效果

请选择 ▼ 请选择 ▼ 请选择 ▼

请选择  
浙江省  
山东省  
广东省  
甘肃省

山东省 ▼ 请选择 ▼ 请选择 ▼

请选择  
济南市  
青岛市  
济宁市  
潍坊市

山东省 ▼ 青岛市 ▼ 青岛一区 ▼

请选择  
青岛一区  
青岛二区

### 2. 代码分析

#### 1. 页面元素

```
<form action="#">
 <select name="prov" id="prov">
 </select>
 <select name="city" id="city">
 </select>
 <select name="area" id="area">
 </select>
</form>
```

#### 2. 初始代码

```
var $prov = $("#prov");
var $city = $("#city");
var $area = $("#area");
$(function () {
 $prov.html("<option value='0'>请选择</option>")
 $city.html("<option value='0'>请选择</option>")
 $area.html("<option value='0'>请选择</option>")
 $.each(data, function (i, e) {
 $prov.append("<option value='" + e.provId + "'>" +
 e.provname + "</option>")
 })
})
```

### 3. 绑定省份

```
$prov.on("change", function () {
 $.each(data, function (i, e) {
 if ($prov.val() == e.provId) {
 $city.html("<option value='0'>请选择</option>");
 $.each(e.citys, function (i, e2) {
 $city.append("<option value='" + e2.cityId + "'>"
 + e2.cityname +
 "</option>")
 })
 }
 })
 if ($prov.val() == 0) {
 $city.html("<option value='0'>请选择</option>");
 }
 if ($city.val() == 0) {
 $area.html("<option value='0'>请选择</option>");
 }
})
```

### 4. 绑定城市

```
$city.on("change", function () {
 $.each(data, function (i, e) {
 if ($prov.val() == e.provId) {
 $.each(e.citys, function (i, e2) {
 if ($city.val() == e2.cityId) {
 $area.html("");
 $.each(e2.areas, function (i, e3) {
 $area.append("<option value='"
 + e3.areaId + "'>" + e3
 .areaname + "</option>")
 })
 }
 })
 }
 })
 if ($city.val() == 0) {
 $area.html("<option value='0'>请选择</option>");
 }
})
```