1.文档流

- 所有的元素默认情况下都是在文档流中存在的
- 文档流是网页的最底层
- 元素在文档流中的特点：
  - 块元素
    1.默认宽度是父元素的全部
    2.默认高度被内容（子元素）撑开
    3.在页面中自上而下垂直排列
  - 内联元素
    1.默认高度和宽度都被内容撑开
    2.在页面中自左向右水平排列，如果一行不足以容下所有的元素
    则换到下一行继续从左向右

2.浮动

- 使用 float 来设置元素的浮动
- 可选值：
  none 默认值，元素不浮动，就在文档流中
  left 元素向页面的左侧浮动
  right 元素向页面的右侧浮动
- 浮动特点：
  1.元素设置浮动以后，会完全脱离文档流，并向页面的左上或右上浮动。
  直到遇到父元素的边框或其他的父元素时则停止浮动。
  2.如果浮动元素上边是一个没有浮动的块元素，元素不会超过该块元素。
  3.浮动元素的浮动位置不能超过他上边浮动的兄弟元素，最多一边齐
  4.浮动元素不会覆盖文字，文字会围绕在浮动元素的周围，所以可以通过浮动来实现文字环绕图片的效果。

- 浮动以后元素会完全脱离文档流，脱离文档流以后元素会具有如下特点：
  1.块元素不独占一行
  2.块元素的宽度和高度都被内容撑开
  3.元素不在文档流占用位置
  4.内联元素会变成块元素

- 高度塌陷
  - 在文档流中元素的高度默认被子元素撑开，当子元素浮动时，子元素会脱离文档流，
  此时将不能撑起父元素的高度，会导致父元素的高度塌陷。父元素高度塌陷会导致其他元素的位置上移，导致页面的布局混乱。
  - 可以通过开启元素的 BFC 来处理高度塌陷的问题。
    - BFC 叫做 Block Formatting Context
    - 它是一个隐含属性，默认情况是关闭，当开启以后元素会具有如下的特性：
      1.父元素的垂直外边距不会和子元素重叠
      2.开启 BFC 的元素不会被浮动元素覆盖
      3.父元素可以包含浮动的子元素 ******

- 开启 BFC 的方式很多：
  1.设置元素浮动
  2.设置元素绝对定位
  3.设置元素为 inline-block
  4.将元素的 overflow 设置为一个非默认值
- 一般我们采取副作用比较小的方式
  overflow:hidden;

3.定位
- 通过定位可以将元素摆放到页面的任意位置
- 使用 position 来设置元素的定位
  - 可选值：
    - static 默认值 元素不开启定位
    - relative 开启元素的相对定位
    - absolute 开启元素的绝对定位
    - fixed 开启元素的固定定位

  - 相对定位
    1.相对于元素自身在文档流中的位置进行定位
    2.相对定位的元素不会脱离文档流，定位元素的性质不会改变，块还是块，内联还是内联
    3.如果不设置偏移量，元素不会发生任何的变化
    4.会提升元素的层级

  - 绝对定位
    1.相对于离它最近的开启了定位的祖先元素进行定位，如果祖先元素都没有开启定位则相对于浏览器窗口进行定位。
    2.绝对定位会使元素完全脱离文档流，会改变元素的性质，内联变成块元素，块元素的宽度被内容撑开
    3.绝对定位的元素如果不设置偏移量，元素的位置不会发生变化
    4.会提升元素的层级

  - 固定定位
    - 固定定位也是一种绝对定位，它的大部分特点都和绝对定位是相同的。
    - 不同的是：
      - 固定定位永远相对于浏览器窗口进行定位
      - 固定定位会固定在浏览器的指定的位置，不会随页面一起滚动

  - 偏移量
    - 当元素开启了定位以后，可以通过四个偏移量来设置元素的位置
      top：相对于定位位置的顶部的偏移量
      bottom：相对于定位位置的底部的偏移量
      left：相对于定位位置的左侧的偏移量
      right：相对于定位位置的右侧的偏移量

- 一般只需要使用两个值即可给元素进行定位
  top left
  top right
  bottom left
  bottom right

- 偏移量也可以指定一个负值，如果是负值则元素会向相反的方向移动

- 层级
  - 当元素开启定位以后，可以通过 z-index 来设置层级，
    它需要一个正整数作为参数，值越大层级越高，层级越高越优先显示
    如果层级一样，则后边的会盖住前边的，父元素永远都不会盖住子元素。

  - 文档流 < 浮动 < 定位

- 元素的透明
  使用 opacity 来设置元素的透明度
  - 需要一个 0-1 之间的值
  - 0 表示完全透明
  - 1 表示完全不透明

  IE8 及以下的浏览器不支持该样式，需要使用如下方式来设置
  filter:alpha(opacity=透明度);
  - 需要一个 0-100 之间的值
  - 0 表示完全透明
  - 100 表示完全不透明

1.表格
- 在网页中可以通过表格来表示一些格式化的数据
- 表格相关的标签
  - <table> 用来创建一个表格
  - <tr> 表示表格中的一行
  - <th> 表示表头中的单元格
  - <td> 表示表格中的单元格
    - 属性：
      colspan 横向的合并单元格
      rowspan 纵向的合并单元格
  - 例子：
    <table>
      <tr>
        <td></td>

```
                <td></td>
            </tr>
            <tr>
                <td></td>
                <td></td>
            </tr>
        </table>
```

- 长表格
  - <thead> 表格的头部
  - <tbody> 表格的主体
    - 注意：如果表格中没有写 thead tbody tfoot，浏览器会自动向 table 中添加一个 tbody
      并且将所有的 tr 都放到 tbody 中，tr 是 tbody 的子元素，不是 table 的子元素
  - <tfoot> 表格的底部

2.表单

- 表单可以将用户的信息提交到服务器中
- <form>
  - 用来创建一个表单
  - 属性：
    action:需要一个服务器地址，提交表单时表单中的内容将会被提交到该地址
- 表单项
  - <input />
    - 它可以根据不同的 type 属性值，生成不同的表单项
    - type="text" 文本框 <input type="text" name="" />
    - type="password" 密码框 <input type="password" name="" />
    - type="radio" 单选按钮 <input type="radio" name="" value="" checked="checked" />
    - type="checkbox" 多选框 <input type="checkbox" name="" value="" checked="checked" />
    - type="submit" 提交按钮 <input type="submit" value="按钮上的文字" />
    - type="reset" 重置按钮 <input type="reset" value="按钮上的文字" />
    - type="button" 普通按钮 <input type="button" value="按钮上的文字" />
  - <select>
    - 下拉列表
    - <select name="">

```
            <option value="" selected="selected"></option>
            <option value=""> </option>
            <option value=""></option>
        </select>
```

- <button>
  - 按钮功能 input 那几个按钮一样，但是它们要灵活一些

```
<button type="submit">按钮的文字</button>
<button type="reset">按钮的文字</button>
<button type="button">按钮的文字</button>
```

3.JavaScript
- JavaScript 负责页面中的的行为。
- 它是一门运行在浏览器端的脚本语言。
- JS 的编写的位置

1.可以编写到标签的指定属性中

```
<button onclick="alert('hello');">我是按钮</button>
<a href="javascript:alert('aaa');">超链接</a>
```

2.可以编写到 script 标签中 *****

```
<script type="text/javascript">
    //编写 js 代码
</script>
```

3.可以将代码编写到外部的 js 文件中，然后通过标签将其引入 *****

```
<script type="text/javascript" src="文件路径"></script>
```

- 输出语句
  - alert("要输出的内容");
    - 该语句会在浏览器窗口中弹出一个警告框
  - document.write("要输出的内容");
    - 该内容将会被写到 body 标签中，并在页面中显示
  - console.log("要输出的内容");
    - 该内容会被写到开发者工具的控制台中
- 基本的语法
  - 注释
    - 单行注释

```
//注释内容
```

    - 多行注释

```
/*
    注释内容
*/
```

  - JS 严格区分大小写
  - JS 中每条语句以分号(;)结尾

- JS 中会自动忽略多个空格和换行，所以我们可以利用空格和换行对代码进行格式化。

- 字面量和变量
  - 字面量
    - 字面量实际上就是一些固定的值，比如 1 2 3 4 true false null NaN "hello"

      字面量都是不可以改变的。
    - 由于字面量不是很方便使用，所以在 JS 中很少直接使用字面量

  - 变量
    - 变量可以用来保存字面量，并且可以保存任意的字面量
    - 一般都是通过变量来使用字面量，而不直接使用字面量，而且也可以通过变量来对字面量进行一个描述
    - 声明变量
      - 使用 var 关键字来声明一个变量

      ```
      var a;
      var b;
      var c;
      ```

    - 为变量赋值

      ```
      a = 1;
      b = 2;
      c = 3;
      ```

    - 声明和赋值同时进行 *****

      ```
      var d = 456;
      var e = 789;
      ```

- 标识符
  - 在 JS 中所有的可以自主命名的内容，都可以认为是一个标识符，

    是标识符就应该遵守标识符的规范。
  - 比如：变量名、函数名、属性名
  - 规范：

    1.标识符中可以含有字母、数字、_、$

    2.标识符不能以数字开头

    3.标识符不能是 JS 中的关键字和保留字

    4.标识符一般采用驼峰命名法

    ```
    xxxYyyZzz
    ```

1.数据类型
- JS 中一共分成六种数据类型
  - String 字符串

    - Number 数值
    - Boolean 布尔值
    - Null 空值
    - Undefined 未定义
    - Object 对象
- 其中基本数据类型有
    - String 字符串
        - JS 中的字符串需要使用引号引起来双引号或单引号都行
        - 在字符串中使用\作为转义字符
            \' ==> '
            \" ==> "
            \n ==> 换行
            \t ==> 制表符
            \\ ==> \

        - 使用 typeof 运算符检查字符串时，会返回"string"

    - Number 数值
        - JS 中所有的整数和浮点数都是 Number 类型
        - 特殊的数字
            Infinity 正无穷
            -Infinity 负无穷
            NaN 非法数字（Not A Number）
        - 其他进制的数字的表示：
            0b 开头表示二进制，但是不是所有的浏览器都支持
            0 开头表示八进制
            0x 开头表示十六进制

        - 使用 typeof 检查一个 Number 类型的数据时，会返回"number"
            （包括 NaN 和 Infinity）

    - Boolean 布尔值
        - 布尔值主要用来进行逻辑判断，布尔值只有两个
        - true 逻辑的真
        - false 逻辑的假
        - 使用 typeof 检查一个布尔值时，会返回"boolean"

    - Null 空值
        - 空值专门用来表示为空的对象，Null 类型的值只有一个
        - null
        - 使用 typeof 检查一个 Null 类型的值时会返回"object"

    - Undefined 未定义
        - 如果声明一个变量但是没有为变量赋值此时变量的值就是 undefined

        - 该类型的值只有一个 undefined
        - 使用 typeof 检查一个 Undefined 类型的值时，会返回"undefined"

- 引用数据类型
    - Object 对象

- 类型转换
    - 类型转换就是指将其他的数据类型，转换为 String Number 或 Boolean
    - 转换为 String
        - 方式一（强制类型转换）：
            - 调用被转换数据的 toString()方法
            - 例子：

```
var a = 123;
a = a.toString();
```

            - 注意：这个方法不适用于 null 和 undefined
              由于这两个类型的数据中没有方法，所以调用 toString()时会报错

        - 方式二（强制类型转换）：
            - 调用 String()函数
            - 例子：

```
var a = 123;
a = String(a);
```

            - 原理：对于 Number Boolean String 都会调用他们的 toString()方法来将其转换为字符串，
              对于 null 值，直接转换为字符串"null"。对于 undefined 直接转换为字符串"undefined"

        - 方式三（隐式的类型转换）：*****
            - 为任意的数据类型 +""
            - 例子：

```
var a = true;
a = a + "";
```

            - 原理：和 String()函数一样

    - 转换为 Number
        - 方式一（强制类型转换）：
            - 调用 Number()函数
            - 例子：

```
var s = "123";
s = Number(s);
```

            - 转换的情况：
              1.字符串 --> 数字
                - 如果字符串是一个合法的数字，则直接转换为对应的数字
                - 如果字符串是一个非法的数字，则转换为 NaN

- 如果是一个空串或纯空格的字符串，则转换为0

2.布尔值 --> 数字
- true转换为1
- false转换为0

3.空值 --> 数字
- null转换为0

4.未定义 --> 数字
- undefined 转换为NaN

- 方式二（强制类型转换）：
  - 调用parseInt()或parseFloat()
  - 这两个函数专门用来将一个字符串转换为数字的
  - parseInt()
    - 可以将一个字符串中的有效的整数位提取出来，并转换为Number
    - 例子：

```
var a = "123.456px";
a = parseInt(a); //123
```

    - 如果需要可以在parseInt()中指定一个第二个参数，来指定进制

  - parseFloat()
    - 可以将一个字符串中的有效的小数位提取出来，并转换为Number
    - 例子：

```
var a = "123.456px";
a = parseFloat(a); //123.456
```

- 方式三（隐式的类型转换）：
  - 使用一元的+来进行隐式的类型转换
  - 例子：

```
var a = "123";
a = +a;
```

  - 原理：和Number()函数一样

- 转换为布尔值
  - 方式一（强制类型转换）：
    - 使用Boolean()函数
    - 例子：

```
var s = "false";
s = Boolean(s); //true
```

    - 转换的情况

字符串 --> 布尔
- 除了空串其余全是true

数值 --> 布尔

- 除了 0 和 NaN 其余的全是 true

null、undefined ---> 布尔
- 都是 false

对象 ---> 布尔
- 都是 true

- 方式二（隐式类型转换）：
  - 为任意的数据类型做两次非运算，即可将其转换为布尔值
  - 例子：

```
var a = "hello";
a = !!a; //true
```

- 运算符
  - 运算符也称为操作符
  - 通过运算符可以对一个或多个值进行运算或操作
  - typeof 运算符
    - 用来检查一个变量的数据类型
    - 语法：typeof 变量
    - 它会返回一个用于描述类型的字符串作为结果

  - 算数运算符

    + 对两个值进行加法运算并返回结果
    - 对两个值进行减法运算并返回结果
    * 对两个值进行乘法运算并返回结果
    / 对两个值进行除法运算并返回结果
    % 对两个值进行取余运算并返回结果

    - 除了加法以外，对非 Number 类型的值进行运算时，都会先转换为 Number 然后在做运算。
    - 而做加法运算时，如果是两个字符串进行相加，则会做拼串操作，将两个字符连接为一个字符串。
    - 任何值和字符串做加法，都会先转换为字符串，然后再拼串

  - 一元运算符
    - 一元运算符只需要一个操作数
    - 一元的+
      - 就是正号，不会对值产生任何影响，但是可以将一个非数字转换为数字
      - 例子：

```
var a = true;
a = +a;
```

- 一元的-
  - 就是负号，可以对一个数字进行符号位取反
  - 例子：

```
var a = 10;
a = -a;
```

- 自增
  - 自增可以使变量在原值的基础上自增 1
  - 自增使用 ++
  - 自增可以使用 前++（++a）后++(a++)
  - 无论是++a 还是 a++都会立即使原变量自增 1
    不同的是++a 和 a++的值是不同的，
    ++a 的值是变量的新值（自增后的值）
    a++的值是变量的原值（自增前的值）

- 自减
  - 自减可以使变量在原值的基础上自减 1
  - 自减使用 --
  - 自减可以使用 前--（--a）后--(a--)
  - 无论是--a 还是 a--都会立即使原变量自减 1
    不同的是--a 和 a--的值是不同的，
    --a 的值是变量的新值（自减后的值）
    a--的值是变量的原值（自减前的值）

1.运算符

逻辑运算符

!

- 非运算可以对一个布尔值进行取反，true 变 false false 边 true
- 当对非布尔值使用!时，会先将其转换为布尔值然后再取反
- 我们可以利用!来将其他的数据类型转换为布尔值

&&

- &&可以对符号两侧的值进行与运算
- 只有两端的值都为 true 时，才会返回 true。只要有一个 false 就会返回 false。
- 与是一个短路的与，如果第一个值是 false，则不再检查第二个值
- 对于非布尔值，它会将其转换为布尔值然后做运算，并返回原值
- 规则：
  1.如果第一个值为 false，则返回第一个值
  2.如果第一个值为 true，则返回第二个值

||

- ||可以对符号两侧的值进行或运算

- 只有两端都是 false 时，才会返回 false。只要有一个 true，就会返回 true。
- 或是一个短路的或，如果第一个值是 true，则不再检查第二个值
- 对于非布尔值，它会将其转换为布尔值然后做运算，并返回原值
- 规则：
    1. 如果第一个值为 true，则返回第一个值
    2. 如果第一个值为 false，则返回第二个值

赋值运算符

=

- 可以将符号右侧的值赋值给左侧变量

+=

- a += 5 相当于 a = a+5
- var str = "hello"; str += "world";

-=

- a -= 5 相当于 a = a-5

*=

- a *= 5 相当于 a = a*5

/=

- a /= 5 相当于 a = a/5

%=

- a %= 5 相当于 a = a%5

关系运算符

- 关系运算符用来比较两个值之间的大小关系的

    >

    >=

    <

    <=

- 关系运算符的规则和数学中一致，用来比较两个值之间的关系，
  如果关系成立则返回 true，关系不成立则返回 false。
- 如果比较的两个值是非数值，会将其转换为 Number 然后再比较。
- 如果比较的两个值都是字符串，此时会比较字符串的 Unicode 编码，而不会转换为 Number。

相等运算符

==

- 相等，判断左右两个值是否相等，如果相等返回 true，如果不等返回 false
- 相等会自动对两个值进行类型转换，如果对不同的类型进行比较，会将其转

换为相同的类型然后再比较，
转换后相等它也会返回 true

!=

- 不等，判断左右两个值是否不等，如果不等则返回 true，如果相等则返回 false
- 不等也会做自动的类型转换。

===

- 全等，判断左右两个值是否全等，它和相等类似，只不过它不会进行自动的类型转换，
如果两个值的类型不同，则直接返回 false

!==

- 不全等，和不等类似，但是它不会进行自动的类型转换，如果两个值的类型不同，它会直接返回 true

特殊的值：

- null 和 undefined
  - 由于 undefined 衍生自 null，所以 null == undefined 会返回 true。
    但是 null === undefined 会返回 false。

- NaN
  - NaN 不与任何值相等，报告它自身 NaN == NaN //false

- 判断一个值是否是 NaN
  - 使用 isNaN()函数

三元运算符：

?:

- 语法：条件表达式?语句 1:语句 2;
- 执行流程：
  先对条件表达式求值判断，
    如果判断结果为 true，则执行语句 1，并返回执行结果
    如果判断结果为 false，则执行语句 2，并返回执行结果

优先级：

- 和数学中一样，JS 中的运算符也是具有优先级的，
  比如 先乘除 后加减 先与 后或
- 具体的优先级可以参考优先级的表格，在表格中越靠上的优先级越高，
  优先级越高的越优先计算，优先级相同的，从左往右计算。
- 优先级不需要记忆，如果越到拿不准的，使用()来改变优先级。

2.流程控制语句

- 程序都是自上向下的顺序执行的，

通过流程控制语句可以改变程序执行的顺序，或者反复的执行某一段的程序。

- 分类：
  1.条件判断语句
  2.条件分支语句
  3.循环语句

条件判断语句

- 条件判断语句也称为 if 语句
- 语法一：

```
if(条件表达式){
    语句...
}
```

- 执行流程：
  if 语句执行时，会先对条件表达式进行求值判断，
  如果值为 true，则执行 if 后的语句
  如果值为 false，则不执行

- 语法二：

```
if(条件表达式){
    语句...
}else{
    语句...
}
```

- 执行流程：
  if...else 语句执行时，会对条件表达式进行求值判断，
  如果值为 true，则执行 if 后的语句
  如果值为 false，则执行 else 后的语句

- 语法三：

```
if(条件表达式){
    语句...
}else if(条件表达式){
    语句...
}else if(条件表达式){
    语句...
}else if(条件表达式){
    语句...
}else{
    语句...
}
```

- 执行流程

- if...else if...else 语句执行时，会自上至下依次对条件表达式进行求值判断，

如果判断结果为 true，则执行当前 if 后的语句，执行完成后语句结束。

如果判断结果为 false，则继续向下判断，直到找到为 true 的为止。
如果所有的条件表达式都是 false，则执行 else 后的语句

1.条件分支语句

- switch 语句
- 语法:

```
switch(条件表达式){
    case 表达式:
        语句...
        break;
    case 表达式:
        语句...
        break;
    case 表达式:
        语句...
        break;
    default:
        语句...
        break;
}
```

- 执行流程：
  - switch...case...语句在执行时，会依次将 case 后的表达式的值和 switch 后的表达式的值进行全等比较，

如果比较结果为 false,则继续向下比较。如果比较结果为 true,则从当前 case 处开始向下执行代码。

如果所有的 case 判断结果都为 false，则从 default 处开始执行代码。

2.循环语句

- 通过循环语句可以反复执行某些语句多次
- while 循环
  - 语法：

```
while(条件表达式){
    语句...
}
```

  - 执行流程：

while 语句在执行时，会先对条件表达式进行求值判断，

如果判断结果为 false，则终止循环

如果判断结果为 true，则执行循环体
循环体执行完毕，继续对条件表达式进行求值判断，依此类推

- do...while 循环
  - 语法:
    ```
    do{
        语句...
    }while(条件表达式)
    ```
  - 执行流程
    do...while 在执行时，会先执行 do 后的循环体，然后在对条件表达式进行判断，
    如果判断判断结果为 false，则终止循环。
    如果判断结果为 true，则继续执行循环体，依此类推
  - 和 while 的区别：
    while：先判断后执行
    do...while: 先执行后判断
    - do...while 可以确保循环体至少执行一次。

- for 循环
  - 语法：
    ```
    for(①初始化表达式 ; ②条件表达式 ; ④更新表达式){
        ③语句...
    }
    ```
  - 执行流程：
    首先执行①初始化表达式，初始化一个变量，
    然后对②条件表达式进行求值判断，如果为 false 则终止循环
    如果判断结果为 true，则执行③循环体
    循环体执行完毕，执行④更新表达式，对变量进行更新。
    更新表达式执行完毕重复②

- 死循环
  ```
  while(true){

  }

  for(;;){

  }
  ```

==========

```
var a = 123;
function fun(){
    alert(a);
}
fun();
```

==========

```
var a = 123;
function fun(){
    alert(a);
    var a = 456;
}
fun();
alert(a);
```

==========

```
var a = 123;
function fun(){
    alert(a);
    a = 456;
}
fun();
alert(a);
```

==========

```
var a = 123;
function fun(a){
    alert(a);
    a = 456;
}
fun();
alert(a);
```

==========

```
var a = 123;
function fun(a){
    alert(a);
    a = 456;
}
fun(123);
```

```
alert(a);
```

1.对象（Object）

- 对象是 JS 中的引用数据类型
- 对象是一种复合数据类型，在对象中可以保存多个不同数据类型的属性
- 使用 typeof 检查一个对象时，会返回 object
- 创建对象
  - 方式一：
    - var obj = new Object();
  - 方式二：
    - var obj = {};

- 向对象中添加属性
  - 语法：

    对象.属性名 = 属性值;

    对象["属性名"] = 属性值;

    - 对象的属性名没有任何要求，不需要遵守标识符的规范，
      但是在开发中，尽量按照标识符的要求去写。
    - 属性值也可以任意的数据类型。

- 读取对象中的属性
  - 语法：

    对象.属性名

    对象["属性名"]
  - 如果读取一个对象中没有的属性，它不会报错，而是返回一个 undefined

- 删除对象中的属性
  - 语法：

    delete 对象.属性名

    delete 对象["属性名"]

- 使用 in 检查对象中是否含有指定属性
  - 语法："属性名" in 对象
    - 如果在对象中含有该属性，则返回 true
      如果没有则返回 false

- 使用对象字面量，在创建对象时直接向对象中添加属性

  语法：

```
var obj = {
                属性名:属性值,
                属性名:属性值,
                属性名:属性值,
                属性名:属性值
```

```
}
```

- 基本数据类型和引用数据类型
  - 基本数据类型
    String Number Boolean Null Undefined
  - 引用数据类型
    Object
  - 基本数据类型的数据，变量是直接保存的它的值。
    变量与变量之间是互相独立的，修改一个变量不会影响其他的变量。
  - 引用数据类型的数据，变量是保存的对象的引用（内存地址）。
    如果多个变量指向的是同一个对象，此时修改一个变量的属性，会影响其他的变量。
  - 比较两个变量时，对于基本数据类型，比较的就是值，
    对于引用数据类型比较的是地址，地址相同才相同

2.函数（Function）
- 函数也是一个对象，也具有普通对象的功能
- 函数中可以封装一些代码，在需要的时候可以去调用函数来执行这些代码
- 使用 typeof 检查一个函数时会返回 function
- 创建函数
  - 函数声明
    ```
    function 函数名([形参 1,形参 2...形参 N]){
        语句...
    }
    ```
  - 函数表达式
    ```
    var 函数名 = function([形参 1,形参 2...形参 N]){
        语句...
    };
    ```
- 调用函数
  - 语法：函数对象([实参 1,实参 2...实参 N]);
    fun() sum() alert() Number() parseInt()
  - 当我们调用函数时，函数中封装的代码会按照编写的顺序执行
- 形参和实参
  - 形参：形式参数
    - 定义函数时，可以在()中定义一个或多个形参，形参之间使用,隔开
      定义形参就相当于在函数内声明了对应的变量但是并不赋值，
      形参会在调用时才赋值。
  - 实参：实际参数
    - 调用函数时，可以在()传递实参，传递的实参会赋值给对应的形参,
      调用函数时 JS 解析器不会检查实参的类型和个数，可以传递任意数据类

型的值。

如果实参的数量大于形参，多余实参将不会赋值，
如果实参的数量小于形参，则没有对应实参的形参将会赋值 undefined

1.函数

- 返回值，就是函数执行的结果。
  - 使用 return 来设置函数的返回值。
  - 语法：return 值;
    - 该值就会成为函数的返回值，可以通过一个变量来接收返回值
  - return 后边的代码都不会执行，一旦执行到 return 语句时，函数将会立刻退出。
  - return 后可以跟任意类型的值，可以是基本数据类型，也可以是一个对象。
  - 如果 return 后不跟值，或者是不写 return 则函数默认返回 undefined。
  - break、continue 和 return
    - break
      - 退出循环
    - continue
      - 跳过当次循环
    - return
      - 退出函数

- 参数，函数的实参也可以是任意的数据类型。

- 方法（method）
  - 可以将一个函数设置为一个对象的属性，
    当一个对象的属性是一个函数时，
    我们称这个函数是该对象的方法。
  - 对象.方法名();
  - 函数名();

2.作用域

- 作用域简单来说就是一个变量的作用范围。
- 在 JS 中作用域分成两种：
  1.全局作用域
    - 直接在 script 标签中编写的代码都运行在全局作用域中
    - 全局作用域在打开页面时创建，在页面关闭时销毁。
    - 全局作用域中有一个全局对象 window，window 对象由浏览器提供，
      可以在页面中直接使用，它代表的是整个的浏览器的窗口。
    - 在全局作用域中创建的变量都会作为 window 对象的属性保存
      在全局作用域中创建的函数都会作为 window 对象的方法保存
    - 在全局作用域中创建的变量和函数可以在页面的任意位置访问。
      在函数作用域中也可以访问到全局作用域的变量。
    - 尽量不要在全局中创建变量

2.函数作用域

- 函数作用域是函数执行时创建的作用域，每次调用函数都会创建一个新的函数作用域。
- 函数作用域在函数执行时创建，在函数执行结束时销毁。
- 在函数作用域中创建的变量，不能在全局中访问。
- 当在函数作用域中使用一个变量时，它会先在自身作用域中寻找，
  如果找到了则直接使用，如果没有找到则到上一级作用域中寻找，
  如果找到了则使用，找不到则继续向上找，一直会

- 变量的声明提前
  - 在全局作用域中，使用 var 关键字声明的变量会在所有的代码执行之前被声明，但是不会赋值。
    所以我们可以在变量声明前使用变量。但是不使用 var 关键字声明的变量不会被声明提前。
  - 在函数作用域中，也具有该特性，使用 var 关键字声明的变量会在函数所有的代码执行前被声明，
    如果没有使用 var 关键字声明变量，则变量会变成全局变量

- 函数的声明提前
  - 在全局作用域中，使用函数声明创建的函数（function fun(){}）,会在所有的代码执行之前被创建，
    也就是我们可以在函数声明前去调用函数，但是使用函数表达式(var fun = function(){})创建的函数没有该特性
  - 在函数作用域中，使用函数声明创建的函数，会在所有的函数中的代码执行之前就被创建好了。

3.this（上下文对象）

- 我们每次调用函数时，解析器都会将一个上下文对象作为隐含的参数传递进函数。
  使用 this 来引用上下文对象，根据函数的调用形式不同，this 的值也不同。
- this 的不同的情况：
  1.以函数的形式调用时，this 是 window
  2.以方法的形式调用时，this 就是调用方法的对象
  3.以构造函数的形式调用时，this 就是新创建的对象

4.构造函数

- 构造函数是专门用来创建对象的函数
- 一个构造函数我们也可以称为一个类
- 通过一个构造函数创建的对象，我们称该对象时这个构造函数的实例
- 通过同一个构造函数创建的对象，我们称为一类对象
- 构造函数就是一个普通的函数，只是他的调用方式不同，
  如果直接调用，它就是一个普通函数
  如果使用 new 来调用，则它就是一个构造函数

- 例子：

```
function Person(){

}
```

- 构造函数的执行流程：
  1.创建一个新的对象
  2.将新的对象作为函数的上下文对象（this）
  3.执行函数中的代码
  4.将新建的对象返回

- instanceof 用来检查一个对象是否是一个类的实例
  - 语法：对象 instanceof 构造函数
    - 如果该对象时构造函数的实例，则返回 true，否则返回 false
    - Object 是所有对象的祖先，所以任何对象和 Object 做 instanceof 都会返回 true

- 枚举对象中的属性
  for...in
  语法：

```
for(var 属性名 in 对象){

}
```

  for...in 语句的循环体会执行多次，对象中有几个属性就会执行几次，
  每次讲一个属性名赋值给我们定义的变量，我们可以通过它来获取对象中的属性

1.原型（prototype）
- 创建一个函数以后，解析器都会默认在函数中添加一个数 prototype
  prototype 属性指向的是一个对象，这个对象我们称为原型对象。
- 当函数作为构造函数使用,它所创建的对象中都会有一个隐含的属性执行该原型对象。
  这个隐含的属性可以通过对象.__proto__来访问。
- 原型对象就相当于一个公共的区域，凡是通过同一个构造函数创建的对象他们通常都可以访问到相同的原型对象。
  我们可以将对象中共有的属性和方法统一添加到原型对象中，
  这样我们只需要添加一次，就可以使所有的对象都可以使用。
- 当我们去访问对象的一个属性或调用对象的一个方法时，它会先自身中寻找，
  如果在自身中找到了，则直接使用。
  如果没有找到，则去原型对象中寻找，如果找到了则使用，
  如果没有找到，则去原型的原型中寻找，依此类推。直到找到 Object 的原型为止，Object 的原型的原型为 null，
  如果依然没有找到则返回 undefined

- hasOwnProperty()
  - 这个方法可以用来检查对象自身中是否含有某个属性
  - 语法：对象.hasOwnProperty("属性名")

2.数组（Array）
- 数组也是一个对象，是一个用来存储数据的对象
  和 Object 类似，但是它的存储效率比普通对象要高
- 数组中保存的内容我们称为元素
- 数组使用索引（index）来操作元素
- 索引指由 0 开始的整数
- 数组的操作：
  - 创建数组
    - var arr = new Array();
    - var arr = [];

  - 向数组中添加元素
    - 语法；
      数组对象[索引] = 值;
      arr[0] = 123;
      arr[1] = "hello";

  - 创建数组时直接添加元素
    - 语法：
      var arr = [元素 1,元素 2....元素 N];
      - 例子：
        var arr = [123,"hello",true,null];

  - 获取和修改数组的长度
    - 使用 length 属性来操作数组的长度
    - 获取长度：
      数组.length
      - length 获取到的是数组的最大索引+1
      - 对于连续的数组，length 获取到的就是数组中元素的个数
    - 修改数组的长度
      数组.length = 新长度
        - 如果修改后的 length 大于原长度，则多出的部分会空出来
        - 如果修改后的 length 小于原长度，则原数组中多出的元素会被删除
    - 向数组的最后添加元素
      数组[数组.length] = 值;

- 数组的方法
  - push()
    - 用来向数组的末尾添加一个或多个元素，并返回数组新的长度

    - 语法：数组.push(元素 1,元素 2,元素 N)
  - pop()
    - 用来删除数组的最后一个元素，并返回被删除的元素
  - unshift()
    - 向数组的前边添加一个或多个元素，并返回数组的新的长度
  - shift()
    - 删除数组的前边的一个元素，并返回被删除的元素
  - slice()
    - 可以从一个数组中截取指定的元素
    - 该方法不会影响原数组，而是将截取到的内容封装为一个新的数组并返回
    - 参数：
      1.截取开始位置的索引（包括开始位置）
      2.截取结束位置的索引（不包括结束位置）
        - 第二个参数可以省略不写，如果不写则一直截取到最后
      - 参数可以传递一个负值，如果是负值，则从后往前数
  - splice()
    - 可以用来删除数组中指定元素，并使用新的元素替换
      该方法会将删除的元素封装到新数组中返回
    - 参数：
      1.删除开始位置的索引
      2.删除的个数
      3.三个以后，都是替换的元素，这些元素将会插入到开始位置索引的前边

- 遍历数组
  - 遍历数组就是将数组中元素都获取到
  - 一般情况我们都是使用 for 循环来遍历数组：

```
for(var i=0 ; i<数组.length ; i++){
    //数组[i]
}
```

  - 使用 forEach()方法来遍历数组（不兼容 IE8）

```
数组.forEach(function(value , index , obj){

});
```

    forEach()方法需要一个回调函数作为参数，
      数组中有几个元素，回调函数就会被调用几次，
      每次调用时，都会将遍历到的信息以实参的形式传递进来，
      我们可以定义形参来获取这些信息。
      value:正在遍历的元素
      index:正在遍历元素的索引
      obj:被遍历对象

1.数组方法

reverse()

- 可以用来反转一个数组，它会对原数组产生影响

concat()

- 可以连接两个或多个数组，它不会影响原数组，而是新数组作为返回值返回

join()

- 可以将一个数组转换为一个字符串
- 参数：

  需要一个字符串作为参数，这个字符串将会作为连接符来连接数组中的元素

  如果不指定连接符则默认使用,

sort()

- 可以对一个数组中的内容进行排序，默认是按照 Unicode 编码进行排序

  调用以后，会直接修改原数组。
- 可以自己指定排序的规则，需要一个回调函数作为参数：

```
function(a,b){

    //升序排列
    //return a-b;

    //降序排列
    return b-a;
}
```

2.函数

- call()
- apply()
  - 这两个方法都是函数对象的方法需要通过函数对象来调用
  - 通过两个方法可以直接调用函数，并且可以通过第一个实参来指定函数中 this
  - 不同的是 call 是直接传递函数的实参而 apply 需要将实参封装到一个数组中传递
- arguments
  - arguments 和 this 类似，都是函数中的隐含的参数
  - arguments 是一个类数组元素，它用来封装函数执行过程中的实参

    所以即使不定义形参，也可以通过 arguments 来使用实参
  - arguments 中有一个属性 callee 表示当前执行的函数对象

- this
  - this 是函数的上下文对象，根据函数的调用方式不同会执向不同的对象

    1.以函数的形式调用时，this 是 window

    2.以方法的形式调用时，this 是调用方法的对象

    3.以构造函数的形式调用时，this 是新建的那个对象

    4.使用 call 和 apply 调用时，this 是指定的那个对象

5.在全局作用域中 this 代表 window

3.包装类

- 在 JS 中为我们提供了三个包装类：

  String() Boolean() Number()

  - 通过这三个包装类可以创建基本数据类型的对象

  例子：

```
var num = new Number(2);
var str = new String("hello");
var bool = new Boolean(true);
```

  - 但是在实际应用中千万不要这么干。

- 当我们去操作一个基本数据类型的属性和方法时，

  解析器会临时将其转换为对应的包装类，然后再去操作属性和方法，

  操作完成以后再将这个临时对象进行销毁。

4.字符串的相关的方法

length

- 获取字符串的长度

charAt()

- 根据索引获取指定的字符

charCodeAt()

- 根据索引获取指定的字符编码

String.fromCharCode()

- 根据字符编码获取字符

indexOf()

lastIndexOf()

- 从一个字符串中检索指定内容
- 需要一个字符串作为参数，这个字符串就是要检索的内容，

  如果找到该内容，则会返回其第一次出现的索引，如果没有找到则返回-1。
- 可以指定一个第二个参数，来表示开始查找的位置
- indexOf()是从前向后找
- lastIndexOf()是从后向前找

slice()

- 可以从一个字符串中截取指定的内容，并将截取到内容返回，不会影响原变量
- 参数：

  第一个：截取开始的位置（包括开始）

  第二个：截取结束的位置（不包括结束）

  - 可以省略第二个参数，如果省略则一直截取到最后
- 可以传负数，如果是负数则从后往前数

substr()

- 和 slice()基本一致，不同的是它第二个参数不是索引，而是截取的数量

substring()

- 和slice()基本一致，不同的是它不能接受负值作为参数，如果设置一个负值，则会自动修正为0，
substring()中如果第二个参数小于第一个，自动调整位置

toLowerCase()
- 将字符串转换为小写并返回

toUpperCase()
- 将字符串转换为大写并返回

split()
- 可以根据指定内容将一个字符串拆分为一个数组
- 参数：
  - 需要一个字符串作为参数，将会根据字符串去拆分数组
    可以接收一个正则表达式，此时会根据正则表达式去拆分数组

match()
- 可以将字符串中和正则表达式匹配的内容提取出来
- 参数：
  - 正则表达式，可以根据该正则表达式将字符串中符合要求的内容提取出来
    并且封装到一个数组中返回

replace()
- 可以将字符串中指定内容替换为新的内容
- 参数：
  - 第一个：被替换的内容，可以是一个正则表达式
  - 第二个：替换的新内容

search()
- 可以根据正则表达式去字符串中查找指定的内容
- 参数：
  正则表达式，将会根据该表达式查询内容，
  并且将第一个匹配到的内容的索引返回，如果没有匹配到任何内容，则返回-1。

5.正则表达式
- 正则用来定义一些字符串的规则，程序可以根据这些规则来判断一个字符串是否符合规则，
  也可以将一个字符串中符合规则的内容提取出来。
- 创建正则表达式
  - var reg = new RegExp("正则","匹配模式");
  - var reg = /正则表达式/匹配模式

- 语法：
  匹配模式：
  i:忽略大小写
  g:全局匹配模式

- 设置匹配模式时，可以都不设置，也可以设置 1 个，也可以全设置，设置时没有顺序要求

正则语法
| 或
[] 或
[^] 除了
[a-z] 小写字母
[A-Z] 大写字母
[A-z] 任意字母
[0-9] 任意数字

- 方法：
test()
- 可以用来检查一个字符串是否符合正则表达式
- 如果符合返回 true，否则返回 false

6.Date
- 日期的对象，在 JS 中通过 Date 对象来表示一个时间
- 创建对象
- 创建一个当前的时间对象
var d = new Date();
- 创建一个指定的时间对象
var d = new Date("月/日/年 时:分:秒");

- 方法：
getDate()
- 当前日期对象是几日（1-31）

getDay()
- 返回当前日期对象时周几（0-6）
- 0 周日
- 1 周一 。。。

getMonth()
- 返回当前日期对象的月份（0-11）
- 0 一月 1 二月 。。。
getFullYear() 从 Date 对象以四位数字返回年份。

getHours() 返回 Date 对象的小时 (0 ~ 23)。
getMinutes() 返回 Date 对象的分钟 (0 ~ 59)。
getSeconds() 返回 Date 对象的秒数 (0 ~ 59)。
getMilliseconds() 返回 Date 对象的毫秒(0 ~ 999)。

getTime()
- 返回当前日期对象的时间戳
- 时间戳，指的是从 1970 年月 1 日 0 时 0 分 0 秒，到现在时间的毫秒数
  计算机底层保存时间都是以时间戳的形式保存的。

Date.now()
- 可以获取当前代码执行时的时间戳

7.Math
- Math 属于一个工具类，它不需要我们创建对象，它里边封装了属性运算相关的常量和方法
  我们可以直接使用它来进行数学运算相关的操作
- 方法：

Math.PI
- 常量，圆周率

Math.abs()
- 绝对值运算

Math.ceil()
- 向上取整

Math.floor()
- 向下取整

Math.round()
- 四舍五入取整

Math.random()
- 生成一个 0-1 之间的随机数
- 生成一个 x-y 之间的随机数
  Math.round(Math.random()*(y-x)+x);

Math.pow(x,y)
- 求 x 的 y 次幂

Math.sqrt()
- 对一个数进行开方

Math.max()
- 求多个数中最大值

Math.min()
- 求多个数中的最小值

1.正则表达式
- 语法：
  - 量词
    {n} 正好 n 次
    {m,n} m-n 次
    {m,} 至少 m 次

+ 至少 1 次 {1,}
? 0 次或 1 次 {0,1}
* 0 次或多次 {0,}

- 转义字符
  \ 在正则表达式中使用\作为转义字符
  \. 表示.
  \\ 表示\
  . 表示任意字符
  \w
    - 相当于[A-z0-9_]
  \W
    - 相当于[^A-z0-9_]
  \d
    - 任意数字
  \D
    - 除了数字
  \s
    - 空格
  \S
    - 除了空格
  \b
    - 单词边界
  \B
    - 除了单词边界
^ 表示开始
$ 表示结束

2.DOM
- Document Object Model
- 文档对象模型，通过 DOM 可以来任意来修改网页中各个内容
- 文档
  - 文档指的是网页，一个网页就是一个文档
- 对象
  - 对象指将网页中的每一个节点都转换为对象
    转换完对象以后，就可以以一种纯面向对象的形式来操作网页了
- 模型
  - 模型用来表示节点和节点之间的关系，方便操作页面
- 节点（Node）
  - 节点是构成网页的最基本的单元，网页中的每一个部分都可以称为是一个节点
  - 虽然都是节点，但是节点的类型却是不同的
  - 常用的节点
    - 文档节点 （Document），代表整个网页
    - 元素节点（Element），代表网页中的标签

- 属性节点（Attribute），代表标签中的属性
- 文本节点（Text），代表网页中的文本内容

- DOM 操作
  - DOM 查询
  - 在网页中浏览器已经为我们提供了 document 对象，
    它代表的是整个网页，它是 window 对象的属性，可以在页面中直接使用。
  - document 查询方法：
    - 根据元素的 id 属性查询一个元素节点对象：
      - document.getElementById("id 属性值");
    - 根据元素的 name 属性值查询一组元素节点对象:
      - document.getElementsByName("name 属性值");
    - 根据标签名来查询一组元素节点对象：
      - document.getElementsByTagName("标签名");

  - 元素的属性：
    - 读取元素的属性：
      语法：元素.属性名
      例子：ele.name
            ele.id
            ele.value
            ele.className

    - 修改元素的属性：
      语法：元素.属性名 = 属性值

    - innerHTML
      - 使用该属性可以获取或设置元素内部的 HTML 代码

- 事件（Event）
  - 事件指的是用户和浏览器之间的交互行为。比如：点击按钮、关闭窗口、鼠标移动。。。
  - 我们可以为事件来绑定回调函数来响应事件。
  - 绑定事件的方式：
    1.可以在标签的事件属性中设置相应的 JS 代码
      例子：
        <button onclick="js 代码。。。">按钮</button>
    2.可以通过为对象的指定事件属性设置回调函数的形式来处理事件
      例子：
        <button id="btn">按钮</button>
        <script>
            var btn = document.getElementById("btn");
            btn.onclick = function(){

```
        };
    </script>
```

- 文档的加载
  - 浏览器在加载一个页面时，是按照自上向下的顺序加载的，加载一行执行一行。
  - 如果将 js 代码编写到页面的上边，当代码执行时，页面中的 DOM 对象还没有加载，
    此时将会无法正常获取到 DOM 对象，导致 DOM 操作失败。
  - 解决方式一：
    - 可以将 js 代码编写到 body 的下边

```
<body>
    <button id="btn">按钮</button>
    <script>
        var btn = document.getElementById("btn");
        btn.onclick = function(){

        };
    </script>
</body>
```

  - 解决方式二：
    - 将 js 代码编写到 window.onload = function(){}中
    - window.onload 对应的回调函数会在整个页面加载完毕以后才执行，
      所以可以确保代码执行时，DOM 对象已经加载完毕了

```
<script>
    window.onload = function(){
        var btn = document.getElementById("btn");
        btn.onclick = function(){

        };
    };

</script>
```

1.DOM 查询
- 通过具体的元素节点来查询
  - 元素.getElementsByTagName()
    - 通过标签名查询当前元素的指定后代元素
  - 元素.childNodes
    - 获取当前元素的所有子节点
    - 会获取到空白的文本子节点

- 元素.children
  - 获取当前元素的所有子元素

- 元素.firstChild
  - 获取当前元素的第一个子节点

- 元素.lastChild
  - 获取当前元素的最后一个子节点

- 元素.parentNode
  - 获取当前元素的父元素

- 元素.previousSibling
  - 获取当前元素的前一个兄弟节点

- 元素.nextSibling
  - 获取当前元素的后一个兄弟节点

innerHTML 和 innerText
- 这两个属性并没有在 DOM 标准定义，但是大部分浏览器都支持这两个属性
- 两个属性作用类似，都可以获取到标签内部的内容，
  不同是 innerHTML 会获取到 html 标签，而 innerText 会自动去除标签
- 如果使用这两个属性来设置标签内部的内容时，没有任何区别的

读取标签内部的文本内容
<h1>h1 中的文本内容</h1>
元素.firstChild.nodeValue

- document 对象的其他的属性和方法
document.all
  - 获取页面中的所有元素，相当于 document.getElementsByTagName("*");

document.documentElement
  - 获取页面中 html 根元素

document.body
  - 获取页面中的 body 元素

document.getElementsByClassName()
  - 根据元素的 class 属性值查询一组元素节点对象
  - 这个方法不支持 IE8 及以下的浏览器

document.querySelector()

- 根据 CSS 选择器去页面中查询一个元素
- 如果匹配到的元素有多个，则它会返回查询到的第一个元素

document.querySelectorAll()
- 根据 CSS 选择器去页面中查询一组元素
- 会将匹配到所有元素封装到一个数组中返回，即使只匹配到一个

2.DOM 修改

document.createElement()
- 可以根据标签名创建一个元素节点对象

document.createTextNode()
- 可以根据文本内容创建一个文本节点对象

父节点.appendChild(子节点)
- 向父节点中添加指定的子节点

父节点.insertBefore(新节点,旧节点)
- 将一个新的节点插入到旧节点的前边

父节点.replaceChild(新节点,旧节点)
- 使用一个新的节点去替换旧节点

父节点.removeChild(子节点)
- 删除指定的子节点
- 推荐方式：子节点.parentNode.removeChild(子节点)

1.DOM 对 CSS 的操作
- 读取和修改内联样式
  - 使用 style 属性来操作元素的内联样式
  - 读取内联样式：
    语法：元素.style.样式名
    - 例子：
      元素.style.width
      元素.style.height
      - 注意：如果样式名中带有-，则需要将样式名修改为驼峰命名法
        将-去掉，然后-后的字母改大写
      - 比如：background-color --> backgroundColor
        border-width ---> borderWidth

  - 修改内联样式：
    语法：元素.style.样式名 = 样式值
    - 通过 style 修改的样式都是内联样式，由于内联样式的优先级比较高，

所以我们通过 JS 来修改的样式，往往会立即生效，
但是如果样式中设置了!important，则内联样式将不会生效。

- 读取元素的当前样式
  - 正常浏览器
    - 使用 getComputedStyle()
    - 这个方法是 window 对象的方法，可以返回一个对象，这个对象中保存着当前元素生效样式
    - 参数：
      1.要获取样式的元素
      2.可以传递一个伪元素，一般传 null
    - 例子：
      获取元素的宽度
      ```
      getComputedStyle(box , null)["width"];
      ```
    - 通过该方法读取到样式都是只读的不能修改

  - IE8
    - 使用 currentStyle
    - 语法：
      元素.currentStyle.样式名
    - 例子：
      box.currentStyle["width"]
    - 通过这个属性读取到的样式是只读的不能修改

- 其他的样式相关的属性
  注意：以下样式都是只读的

  clientHeight
    - 元素的可见高度，指元素的内容区和内边距的高度

  clientWidth
    - 元素的可见宽度，指元素的内容区和内边距的宽度

  offsetHeight
    - 整个元素的高度，包括内容区、内边距、边框

  offfsetWidth
    - 整个元素的宽度，包括内容区、内边距、边框

  offsetParent
    - 当前元素的定位父元素
    - 离他最近的开启了定位的祖先元素，如果所有的元素都没有开启定位，则返回 body

  offsetLeft

  offsetTop
    - 当前元素和定位父元素之间的偏移量
    - offsetLeft 水平偏移量 offsetTop 垂直偏移量

scrollHeight
scrollWidth
- 获取元素滚动区域的高度和宽度

scrollTop
scrollLeft
- 获取元素垂直和水平滚动条滚动的距离

判断滚动条是否滚动到底
- 垂直滚动条
  scrollHeight - scrollTop = clientHeight

- 水平滚动
  scrollWidth - scrollLeft = clientWidth

2.事件（Event）
- 事件对象
- 当响应函数被调用时，浏览器每次都会将一个事件对象作为实参传递进响应函数中，这个事件对象中封装了当前事件的相关信息，比如：鼠标的坐标，键盘的按键，鼠标的按键，滚轮的方向。。
- 可以在响应函数中定义一个形参，来使用事件对象，但是在 IE8 以下浏览器中事件对象没有做完实参传递，而是作为 window 对象的属性保存
  - 例子：

```
元素.事件 = function(event){
    event = event || window.event;

};

元素.事件 = function(e){
    e = e || event;

};
```

- 事件的冒泡（Bubble）
- 事件的冒泡指的是事件向上传导，当后代元素上的事件被触发时，将会导致其祖先元素上的同类事件也会触发。
- 事件的冒泡大部分情况下都是有益的，如果需要取消冒泡，则需要使用事件对象来取消
- 可以将事件对象的 cancelBubble 设置为 true，即可取消冒泡
  - 例子：

```
元素.事件 = function(event){
    event = event || window.event;
    event.cancelBubble = true;
};
```