title：html+css+js	author：songyx	date：2022/02/10

1. HTML

基本结构

乱码：编码与解码所采用的字符集不同。

```
<!-- 文档声明用于告诉浏览器当前网页的版本 -->
<!DOCTYPE html>
<!-- 规定网页的语言 -->
<html lang="zh">
    <head>
        <!-- 告知浏览器网页的字符集，避免乱码 -->
        <meta charset="utf-8">
        <!-- 搜索引擎通过title判断网页主要内容，搜索结果的超链接 -->
        <title>网页的标题</title>
    </head>
    <!-- 网页的主体 -->
    <body></body>
</html>
```

实体

用于书写特殊符号（不换行空格：` `、大于：`&gt`、小于：`&lt`）。

meta

```
<meta name="keywords" content="网站的关键字">
<meta name="description" content="搜索引擎的搜索结果">
```

block、inline

行内元素一般放在块元素内部。`<p>`中不能放入任何块元素。`<a>`中可以放入除自身外的任何元素。

块元素总会独占一行，即使规定了宽和高。默认宽度为父元素全部，默认高度为子元素最高高度，垂直方向会发生外边距合并。

行内元素不支持设置宽度和高度，垂直方向的内边距、外边距、边框并不影响布局。

a

```
<a href="" target="_self">默认值，在当前页面中打开链接</a>
<a href="" target="_blank">在新标签页中打开链接</a>
<a href="#">回到当前页面的顶部</a>
<a href="#id">跳转到指定元素的位置</a>
```

img

```
<img src="" alt="图片描述，加载失败时显示，搜索引擎通过该字段查询">
```

audio、video

```
<!-- 音频（可控、自动播放、循环播放） -->
<audio src="" controls autoplay loop></audio>
<!-- 浏览器不兼容时可显示替换文本 -->
<audio controls>
    <source src="" type="">
    请升级浏览器
</audio>
```

`<video>`使用方式基本与`audio`相同。

2. CSS

CSS样式表

多文件引用该样式表，由于浏览器的缓存机制，只需加载一次。

选择器

类型	说明
*	通配选择器，选中页面的全部元素
(element1element2)	交集选择器
(element1，element2)	并集选择器
('空格'，'>'，'+'，'~')	关系选择器，祖宗与后代、父与子、临近兄弟、普通兄弟
[属性名]，[属性名=属性值]，[属性名^=属性值]，[属性名$=属性值]，[属性名*=属性值]	属性选择器，指定值、指定值开头、指定值结尾、指定值为子串

伪类

描述一个元素的特殊状态。

排序伪类：`:nth-child(1)`(所有元素排序)、`:nth-of-type`(同类型元素排序)。

否定伪类：`:not(selector)`，如`:not(:nth-of-type(3))`(除了同类型的第3个元素)。

超链接伪类：`:link`(未访问)、`:visited`(已访问、只能修改颜色)、`:hover`(鼠标悬停)、`:active`(已选择)。hover 必须在 CSS 定义中的 link 和 visited 之后，才能生效。active 必须在 CSS 定义中的 hover 之后才能生效。伪类名称对大小写不敏感。

伪元素

设置元素指定部分的样式。`::before`与`::after`中必须包含`content`。

优先级

内联>id>类和伪类>元素>*>继承的样式。背景相关的，布局相关的样式等无法被继承。

背景相关的，布局相关的样式等无法被继承（inherited）。选择器优先级：内联>id>类和伪类>元素>*>继承的样式

```
.son {
    /* 获取到最高的优先级 */
    background-color: red !important;
}
```

em、rem

em（1em=1fontSize）是相对于字体大小来计算，rem相对于根元素的字体大小。常用于移动端大小适配。

盒子模型

外边距合并问题、盒子模型

```
.son {
    /* 水平偏移量，垂直偏移量，阴影的模糊半径，阴影颜色 */
    box-shadow: 10px -20px 5px rgba(0, 0, 0, 0.2);
    /* 圆形 */
    border-radius: 50%;
}
```

浮动

脱离文档流的特点：不独占一行、宽高可设置、默认宽高取决于内容（无论块元素与行内元素）。

清除浮动：清除浮动元素对当前元素的影响。原理：设置清除浮动后，浏览器会自动为元素添加上外边距。

```
<html>
<body>
    <div class="father"></div>
    <div class="uncle"></div>
</body>
</html>
<style>
    .father {
        float: left;
        width: 200px;
        height: 200px;
        background-color: red;
    }
    .uncle {
        /* 元素未上移 */
        clear: left;
        width: 300px;
        height: 300px;
        background-color: skyblue;
    }
</style>
```

高度塌陷：浮动布局中，父元素高度默认由子元素撑开。当子元素浮动后，父元素高度丢失，父元素之后的元素上移。

```
<html>
<body>
    <div class="father">
```

```
        <div class="son"></div>
    </div>
    <div class="uncle"></div>
</body>
</html>
<style>
    .father {
        border: 10px orange solid;
    }
    .son {
        float: left;
        width: 200px;
        height: 200px;
        background-color: red;
    }
    .uncle {
        width: 300px;
        height: 300px;
        background-color: skyblue;
    }
</style>
```

BFC：块级格式化环境。

开启BFC的元素的特征：不会被浮动元素所覆盖、与父元素外边距不重叠、可包含浮动的子元素。

```
.father {
    border: 10px orange solid;
    /* 父元素不可见宽消失，下方元素上移 */
    float: left;
    /* 父元素不可见宽消失，下方元素未上移 */
    display: inline-block;
    /* 最佳方式 */
    overflow: hidden;
}
```

除启动BFC外，最完美的方式为（伪类+清除浮动）。

```
/* 既可以解决高度塌陷（after+全部三行），又可以解决外边距重叠（before+前两行） */
.father::before,
.father::after {
    content: '';
    display: table;
    clear: both;
}
```

定位

包含块（containing block）：离自己最近的开启了定位的祖先元素，初始包含块为`<HTML>`。

绝对定位元素相对于其包含块进行定位，因此相对定位常用于作为绝对定位的参考系使用。

```
<html>
<body>
    <div class="father">
        <div class="son"></div>
```

```
    </div>
</body>
</html>
<style>
    .father {
        width: 400px;
        height: 400px;
        background-color: orange;
        position: relative;
    }
    .son {
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
        /* 水平居中 */
        margin: auto;
        /* 垂直居中 */
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
    }
</style>
```

字体

字体类别：`serif`（衬线字体）、`sans-serif`（非衬线字体）、`monospace`（等宽字体）。一般放于`font-family`最后，用于兜底。

单行文字垂直居中：`line-height=height`。

字体的垂直对齐：`vertical-align`（top、bottom、middle），也可用于元素垂直居中（不常用）。

```
<html>
<body>
    <div class="father">
        <div class="son"></div>
    </div>
</body>
</html>
<style>
    .father {
        width: 400px;
        height: 400px;
        background-color: orange;
        /* 垂直居中 */
        display: table-cell;
        vertical-align: middle;
    }
    .son {
        width: 100px;
        height: 100px;
        background-color: red;
        /* 水平居中 */
        margin: 0auto;
    }
</style>
```

文字溢出时显示为省略号，四种属性缺一不可。

```
.father {
    width: 100px;
    /* 空白处理：不换行 */
    white-space: nowrap;
    overflow: hidden;
    /* 文字溢出显示方式：省略号 */
    text-overflow: ellipsis;
}
```

`text-decoration`用于设置或删除文本装饰。`text-transform`用于指定文本中的大写和小写字母。`text-shadow`为文本添加阴影，依次为水平阴影、垂直阴影、模糊效果、阴影颜色。`font-style`主要用于指定斜体文本。

```
p {
    text-decoration: underline;
    text-transform: uppercase;
    text-shadow: 2px 2px 5px red;
    font-style: italic;
}
```

背景

```
body {
    background-image: url("tree.png");
    /* 不重复（no-repeat）、水平重复（repeat-x）、垂直重复（repeat-y） */
    background-repeat: no-repeat;
    /* 不随页面滚动（fixed）、随页面滚动（scroll） */
    background-attachment: fixed;
    /* 铺满元素（cover）、在元素中完整显示图片（contain），类似于img中的object-fit */
    background-size: contain;
    /* 指定位置（水平偏移、垂直偏移），常用于雪碧图。也可以指定方位：左上（left top） */
    background-position: 50px 100px;
}
```

宽度

`max-width`窗口缩小至指定宽度内时，元素宽度随着窗口大小变化。`min-width`窗口缩小至指定宽度内时，元素宽度保持不变，出现水平滚动条。

transition

鼠标悬停时元素的显示效果。

```
.div {
    /* 属性名、持续时间、速度曲线、延迟时间 */
    transition: property duration timing-function delay;
}
```

animation

```
div {
    /* 关键帧、持续时间、速度曲线、延迟时间、播放次数（数值n、无限infinite）、是否反向播放（默认否normal、是alternate） */
    animation: name duration timing-function delay iteration-count direction;
}
```

动画实现：雪碧图宽度为456px，包括四个动作。因此宽度为114px、速度曲线为steps(4)。

```
<html>
<body>
    <div class="box"></div>
</body>
</html>
<style>
    .box {
        width: 114px;
        height: 138px;
        background-image: url('./sprite.jpg');
        animation: example 1s steps(4) infinite;
    }
    @keyframes example {
        from {
            background-position: 0 0;
        }
        to {
            background-position: -456px 0;
        }
    }
</style>
```

transform

`transform: translate(50px,100px)`从其当前位置向右移动50个`px`，并向下移动100个`px`。

绝对定位使元素水平垂直居中存在的问题：子元素若未定义宽高，宽高通过内部元素填充获得，则该方法失效。但可以通过转换进行优化。

```
<html>
<body>
    <div class="father">
        <div class="son">通过内部元素填充宽高</div>
    </div>
</body>
</html>
<style>
    .father {
        width: 400px;
        height: 400px;
        background-color: orange;
        position: relative;
    }
    .son {
        background-color: red;
        position: absolute;
```

```
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
    }
</style>
```

`rotateX()` 绕 `x` 轴旋转。`rotateY()` 绕 `y` 轴旋转。`rotateZ()` 括号内为正则顺时针旋转，括号内为负则逆时针旋转。

实现时钟动画效果。

```
<html>
<body>
    <div class="panel">
        <div class="second-box">
            <div class="second"></div>
        </div>
    </div>
</body>
</html>
<style>
    .panel {
        width: 300px;
        height: 300px;
        margin: 100px auto;
        border: 5px solid black;
        border-radius: 50%;
        position: relative;
    }
    .panel > div {
        margin: auto;
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
    }
    .second-box {
        width: 100%;
        height: 100%;
        animation: run 60s steps(60)infinite;
    }
    .second {
        width: 2px;
        height: 50%;
        margin: 0 auto;
        background-color: red;
    }
    @keyframes run {
        from {
            transform: rotateZ(0);
        }
        to {
            transform: rotateZ(360deg);
        }
    }
</style>
```

`scale(2,3)`元素增大为其原始宽度的两倍和其原始高度的三倍，可用于实现按钮、图片放大。

```
<html>
<body>
    <div class="box">BUY!</div>
</body>
</html>
<style>
    .box {
        margin: 100px;
        width: 60px;
        font-size: 20px;
        background-color: #d2e395;
        border-radius: 10px;
        text-align: center;
        vertical-align: middle;
        transition: transform 2s, background-color 1s;
        cursor: pointer;
        /* 指定变形的原点，默认为center */
        transform-origin: 0 0;
    }
    .box:hover {
        transform: scale(2, 2);
        background-color: #22c9e2;
    }
</style>
```

flex

容器相关：

`flex-direction`定义父元素要在哪个方向上堆叠 flex 项目，依次为：从左到右、从右到左、从上到下、从下到上。

`flex-wrap`规定是否应该对 flex 项目换行，依次为：默认不换行、必要时进行换行、相反顺序换行。

```
.box {
    display: flex;
    flex-direction: row | row-reverse | column | column-reverse;
    flex-wrap: nowrap | wrap | wrap-reverse;
}
```

`justify-content`定义了项目在主轴（水平方向）上的对齐方式，依次为：默认左对齐、右对齐、居中、项目之间的间隔都相等、每个项目两侧的间隔相等。

```
.box {
    display: flex;
    justify-content: flex-start | flex-end | center | space-between | space-
around;
}
```

`align-items`定义项目在交叉轴（竖直方向）上的对齐方式，依次为：默认占满整个容器的高度、交叉轴的起点对齐、交叉轴的终点对齐、交叉轴的中点对齐、项目的第一行文字的基线对齐。

```
.box {
    display: flex;
    align-items: stretch | flex-start | flex-end | center | baseline;
}
```

由于换行产生了多条水平轴线，`align-content`定义了多根轴线的对齐方式，依次为：与交叉轴的起点对齐、与交叉轴的终点对齐、与交叉轴的中点对齐、轴线之间的间隔平均分布、每根轴线两侧的间隔都相等、默认轴线占满整个交叉轴。

```
.box {
    display: flex;
    flex-wrap: wrap align-content: flex-start | flex-end | center | space-
between | space-around | stretch;
}
```

项目相关：

`order` 定义项目的排列顺序。数值越小，排列越靠前，默认为0。

`flex-grow` 规定某个项目相对于其余项目将增长多少，默认为0。

`flex-shrink` 规定某个项目相对于其余项目将收缩多少，默认为1。如果一个项目的 flex-shrink 属性为0，其他项目都为1，则空间不足时，前者不缩小。

`flex-basis`规定项目的初始长度，默认值为`auto`，即项目的本来大小。

`flex: 1`或者`flex: auto`相当于`flex-grow: 1; flex-shrink: 1; flex-basis: auto`。

```
.item {
    width: 200px;
    height: 200px;
    order: 2;
    flex: 1 1 auto;
}
```

`align-self`允许单个项目有与其他项目不一样的对齐方式，可覆盖`align-items`。默认值为`auto`，表示继承父元素的`align-items`属性，如果没有父元素，则等同于`stretch`。

```
.item {
    align-self: auto | flex-start | flex-end | center | baseline | stretch;
}
```

viewport

默认情况下，移动端的网页都会将视口`viewport`设置为980像素。如果网页设置宽度超过了980像素，移动端则会对网页进行缩放以完整显示网页。

```
<!-- 将网页视口设置为完美视口，无该设置则为默认情况，移动端适配必须设置完美视口 -->
<meta name="viewport" content="width=device-width,initial-scale=1.0">
```

移动端设计图纸宽度为750px，通过`vw`适配，(100vw/750px)*100=13.333，为了方便编写，可借助`less`。

```
<html lang="zh-CN">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
</head>
<body>
    <div class="box"></div>
</body>
<style>
    html {
        font-size: 13.333vw;
    }
    .box {
        width: 7.5rem;
        height: 100px;
        background-color: orange;
    }
</style>
```

3. LESS

变量

变量的使用遵循就近原则。

```
@basewidth: 100px;
@basewidth: 200px;
@mkdir: myImgs;

div {
    width: @basewidth;
    height: 200px;
    line-height: $height;
    background-image:url('/@{mkdir}/img1.jpg');
}
```

嵌套

`&`表示当前选择器的父级。

```
.box {
    color: black;
    >a {
        color: red;
        &:hover {
            color: orange;
        }
    }
    &-subBox {
        color: pink;
    }
}
```

extend

```
.father {
    width: 100px;
    height: 100px;
}
.mother {
    color: red;
    font-size: 16px;
}
.son:extend(.father, .mother) {
    margin: 0 auto;
}
```

mixins

```
.father {
    color: red;
}
.son {
    .father;
    margin: 0 auto;
}
```

混合函数可以定义参数，并可设置参数默认值。

```
.standard-box(@width, @height, @backgroundColor: red) {
    width: @width;
    height: @height;
    background-color: @backgroundColor;
}
.box {
    margin: 0 auto;
    .standard-box(100px, 200px);
}
```

运算

算术运算符`+`、`-`、`*`、`/`可以对任何数字、颜色或变量进行运算。计算的结果以最左侧操作数的单位类型为准。

导入

使用其他文件中的变量与函数，如果导入的文件是`.less`扩展名，则可以将扩展名省略掉。

4. JS

变量

声明的变量不能以数字开头，采用驼峰式命名法。仅声明的变量，其值与类型为`undefined`。

`typeof`用于得到变量的类型，`null`是`Object`对象。

浮点运算可能得到不精确的结果。

`null`和`undefined`没有`toString()`方法。

`parseInt`和`parseFloat`可以得到字符串中有效的整数和小数。

```
// 输出为233
parseInt('233str')
// 输出为NaN（Not a Number）
parseInt('str233str')
```

`&&`第一个值为`true`则返回第二个值；第一个值为`false`，直接返回第一个。

`||`第一个值为`false`则返回第二个值；第一个值为`true`，直接返回第一个。

`NaN`不和任何值相等，包括本身。可以通过`isNaN(n)`判断。

`===`比较时不会做类型转换，类型不同则直接返回`false`。

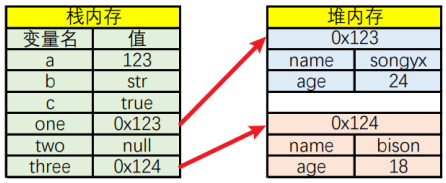
```
// true
'1' == 1
// false
'1' === 1
```

对象

```
var obj = new Object()
// 增、改
obj.name = 'songyx'
obj.age = 24
obj.address = 'chengdu'
console.log(obj)
// 删
delete obj.address
// 检查对象是否包含指定属性名
console.log('address' in obj)
```

栈内存主要用于存储各种基本类型（`Boolean`、`Number`、`String`、`Undefined`、`Null`）的变量以及对象的引用。

堆内存存储对象的属性值。




```
var one = { name: 'songyx' }
var two = {}
// 栈内存中，拷贝对象的引用
two = one
two.name = 'bison'
// {name: 'bison'}
console.log(one)
// 清除对象的引用
two = null
// {name: 'bison'}
console.log(one)
```

`==`在比较对象时，比较的是栈内存中对象的引用是否相同。

遍历对象的属性值与属性名。

```
var obj = {
    name: 'songyx',
    age: 24
}
for (let n in obj) {
    console.log('属性名: ' + n)
    console.log('属性值: ' + obj[n])
}
```

函数

`arguments`中存储实参。

```
function add() {
    console.log(arguments)
}
// arguments[0] = 2、arguments[1] = 3
add(2, 3)
```

函数是一种特殊的对象，通过`()`进行调用。

```
var obj = {
    name: 'songyx',
    age: 24,
    fun: function () {
        var money = 0
        return function () {
            console.log('my money is ' + money)
        }
    }
}
// 调用方式
obj.fun()()
```

调用函数是对值的拷贝，形参拷贝实参的值。这个值可以是一般数据，也可以是地址数据。

作用域

全局作用域：直接编写在`<script>`之中的代码。在全局作用域中无法访问函数作用域的变量。

	声明提前（全局或函数作用域所有代码之前）	**未声明提前（需在声明后使用）**
变量	var a = 10	a = 10
函数	function add(a, b) { console.log(a + b) }	var divide = function (a, b) { console.log(a - b) }

函数作用域在函数定义时确定，函数作用域中使用变量时，从自身开始往上一级作用域寻找，直至全局作用域。

```
// undefined
console.log(c)
var c = 30
function fun() {
    // 30
    console.log(c)
}
fun()
```

this

以函数形式调用，`this`为`window`；以方法形式调用，`this`为调用方法的对象。

```
function fun() {
    console.log(this)
}
var obj = {
    name: 'songyx',
    objFunction: fun
}
// this is window
fun()
// this is obj
obj.objFunction()
```

`apply()`和`call()`可以指定`this`。

```
var one = {
    name: 'songyx'
}
function add(a, b) {
    console.log(this)
    console.log(a + b)
}
// 或add.apply(one, [2, 3]), this为one
add.call(one, 2, 3)
```

原型对象

应将对象中共有的内容设置到原型对象中。

```
// 构造函数
function Person(name, age, address) {
    this.name = name
    this.age = age
    this.address = address
}
// 避免创建对象时函数被重复创建
Person.prototype.printName = function (params) {
    console.log('my name is ' + this.name)
}
// 覆盖Object.prototype中的toString方法
Person.prototype.toString = function () {
    console.log('name:' + this.name + ' age:' + this.age + ' address:' +
this.address)
}
```

```
var my = new Person('songyx', 24, 'chengdu')
my.printName()
```

垃圾回收

```
var obj = new Object()
// 对象在堆内存中的存储就成了垃圾，js自动回收
obj = null
```

数组

元素可以为任意数据类型。

```
var arr = [[1, '2', true], null, undefined, { name: 'songyx', age: 24 }]
```

```
// 第三个参数是数组本身
arr.forEach(function (value, index, arr) {
    console.log('第一个参数是值：' + value + ' 第二个参数是索引：' + index)
})
```

```
// 不改变原数组，截取区间为[start, end)
var newArr = arr.slice(2, 4)
// 截取区间为[start, arr.length-1]
var newArr = arr.slice(2)
// 截取区间为[start, arr.length-3)
var newArr = arr.slice(1,-3)
/* 改变原数组，返回值为被删除的元素
   第一个元素为开始位置，第二个元素为删除个数
   第三个元素及以后，按顺序将新元素添加到开始位置前 */
arr.splice(1, 2, 'str1', 'str2')
```

正则表达式

字符串的一些函数可将正则表达式作为入参。

```
// ['1', '2', '3', '4', '5', '6', '7', '8', '9']
var arr = '1a2B3c4D5e6F7g8H9'.split(/[A-z]/)
// 7
var index = 'acc|bc|aec'.search(/a[b,e]c/)
/* ['a', 'B', 'c', 'D', 'e', 'F', 'g', 'H']
    i为忽略大小写，g为全局匹配
    若不写g，返回为a */
var arr2 = '1a2B3c4D5e6F7g8H9'.match(/[a-z]/ig)
// '1|2|3|4|5|6|7|8|9'
var str = '1a2B3c4D5e6F7g8H9'.replace(/[a-z]/ig, '|')
```

DOM

```
// 该方法可被其他节点调用
var arr = document.getElementsByTagName('div')
// false，返回值为nodeList，获取其中的元素arr[1]
console.log(Array.isArray(arr))
// <input name="goods" class="search-inp">
var arr2 = document.getElementsByName('goods')
// 获取样式类的名称
console.log(arr2[0].className)
// 参数为CSS选择器，IE8及以上可以使用
var obj = document.querySelector('.container a')
var arr = document.querySelectorAll('.container ~ div')
```

```
var person = document.getElementById('person')
var age = document.getElementById('age')
// 增
var address = document.createElement('li')
address.innerHTML = 'chengdu'
person.appendChild(address)
// 删
person.removeChild(age)
// 改
person.replaceChild(address, age)
// 插
age.parentNode.insertBefore(address, age)
```

event

超链接的`onclick`事件，可以通过`return false`避免跳转。

`IE8`中，浏览器不会传递事件对象，事件对象作为`window`的属性保存。

```
document.onmousemove = function (event) {
    event = event || window.event
    // 鼠标指针相对于可见窗口的水平坐标
    console.log(event.clientX)
}
```

事件的冒泡：触发子元素绑定事件，同时也会触发父元素的绑定事件。

```
// 取消冒泡
event.cancelBubble = true
```

事件的委派：借助事件的冒泡，减少事件绑定的次数，将事件绑定给父元素。

```
<html>
<body>
    <ul id="person" style="background-color: red;width: 100px;height: 100px;">
        <li id="name">songyx</li>
        <li id="age">24</li>
    </ul>
</body>
</html>
<script>
    document.getElementById('person').onclick = function (event) {
        var clickObj = event.target
```

```
        clickObj.style.backgroundColor = 'orange'
    }
</script>
```

事件的绑定

```
// 事件绑定，例如参数为：<div>测试</div>、'click'、function() { console.log(this) }
function bind(element, eventStr, callback) {
    if (element.addEventListener) {
        element.addEventListener(eventStr, callback, false)
    } else {
        // IE8及以下
        element.attachEvent("on" + eventStr, function () {
            // 借助匿名函数和call()，设定回调函数的this为element（之前为window）
            callback(element)
        })
    }
}
```

BOM

`navigator`包含了浏览器的相关信息，`userAgent`属性可用于区分浏览器类型。

```
if (/firefox/i.test(navigator.userAgent)) {
    console.log('firefox')
} else if (/chrome/i.test(navigator.userAgent)) {
    console.log('chrome')
} else if (/msie/i.test(navigator.userAgent)) {
    console.log('IE10及以下')
} else if ('ActiveXObject' in window) {
    console.log('IE11')
}
```

`history`记录了浏览器的历史信息。

```
document.getElementById('btn').onclick = function () {
    // forward() = go(1)、back() = go(-1)
    history.go(n)
}
```

`location`包含了地址栏信息。

```
// 跳转到指定页面，会生成历史记录
location = 'newUrl'
// 参数可选，若为ture则是清除缓存加载页面
location.reload(true)
// 跳转到指定页面，但不会生成历史记录
location.replace('newUrl')
```

`setInterval`返回值为定时器的标识。

```
var flag
document.getElementById('btn').onclick = function () {
    // 避免定时器被重复开启
    clearInterval(flag)
    var index = 0
    flag = setInterval(function () {
        console.log(index++)
        if (index > 5) {
            clearInterval(flag)
        }
    }, 500)
}
```

`setTimeout`只执行一次，返回值为延时调用的标识。

类

采用设置类的方式改变样式，浏览器只需渲染一次。

```
obj.className += ' ' + 'pointer-active'
```

JSON

`JSON`是特殊格式的字符串，属性名必须加双引号，用于前后端数据传输。

```
// 前端接收数据
var myJSON = '{"name":"songyx","age":24}'
var myObj = JSON.parse(myJSON)
// 适配低版本IE浏览器，但执行性能较差且具有安全隐患。
var myObj = eval('(' + myJSON + ')')
```

```
// 前端发送数据
var myObj = { name: 'songyx', age: 24 }
var myJSON = JSON.stringify(myObj)
```

5. JS高阶

`undefined`和`null`的区别?

`undefined`为声明了但未赋值，`null`为声明了且赋值为`null`。

何时赋值为`null`?

1.初始赋值为null，表明之后将要赋值为对象。

2.对象最后赋值为null，利用垃圾回收释放内存。

内存空间自动释放和垃圾回收的区别?

```
function fun() {
    var A = {}
}
/*
 * 函数调用结束后
 * 局部变量A占用的内存空间（栈内存）自动释放。
 * A指向的对象在之后的某个时刻被gc回收。
 */
fun()
```

何时使用`['属性名']`方式?

变量名不确定的时候。

```
var person = {}
var variable = 'name'
person[variable] = 'songyx'
```

回调函数的特征？常见的回调函数?

1.你定义的，你没有调用，但最终它执行了。

2.`dom`事件回调函数、定时器回调函数、`ajax`请求回调函数、生命周期回调函数。

什么是IIFE?

立即执行函数可创建独立的作用域，作用为隐藏实现、不会污染外部命名空间。

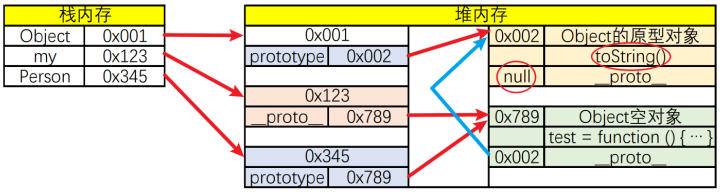
```
console.log('当语句均省略分号时，在IIFE前添加分号避免报错')
; (function () {
    var str = 'testStr'
    function test() {
        console.log(str)
    }
    window.$ = function () {
        return {
            // 通过暴露出的方法访问隐藏变量str
            fun: test
        }
    }
})()
// 输出testStr
$().fun()
```

隐式原型与显式原型的关系？什么是隐式原型链?

显式原型是定义构造函数时自动添加，默认为空`Object`实例对象。

隐式原型是通过构造函数创建实例时添加，构造函数包含隐藏语句：`this.__proto__ == Person.prototype`。

```
var my = new Person('songyx', 24, 'chengdu')
// 显式原型为Person.prototype，隐式原型为my.__proto__，返回为true
console.log(Person.prototype === my.__proto__)
```



当访问对象的属性时。若自身查找不到，则沿着`__proto__`这条链向上查找。若仍然查找不到，返回`undefined`。Object的原型对象为原型链的尽头，因为`Object.prototype.__proto__ = null`。

在显式原型中定义的方法`test()`。通过实例`my`进行调用时，在自身的属性中未查找到，则去查找自身隐式原型的属性，由于隐式原型与显式原型的内在关系，保证一定可以查到。

```
function F() { }
Object.prototype.a = function () {
    console.log('a')
}
Function.prototype.b = function () {
    console.log('b')
}
var f = new F()
// a
f.a()
// 报错
f.b()
// a
F.a()
// b
F.b()
// true(特殊)
console.log(Function.prototype === Function.__proto__)
```

从原型链的角度解释`instanceof`?

`A instanceof B`，当B的显式原型在A的原型链上，就返回true。

变量声明提升与函数声明提升如何产生的?

执行全局代码前对全局数据进行预处理：

1.`var`声明的变量赋值为`undefined`，添加为`window（全局执行上下文）`的属性。

2.`function`赋值为`fun()`，添加为`window`的方法。

3.`this`赋值为`window`。

调用函数前对局部数据进行预处理：

1.形参赋值为实参、`arguments`赋值为实参列表、`var`声明的变量赋值为`undefined`，添加为函数执行上下文（虚拟的，存在于栈内存中）的属性。

2.`function`赋值为`fun()`，添加为函数执行上下文的方法。

3.`this`赋值为调用函数的对象。

函数执行上下文以栈的形式存储，栈的最底层为`window`。开始调用时压栈，结束调用时出栈。

变量声明提升与函数声明提升的优先级?

先执行变量声明提升。

什么是闭包?常见的闭包?

内部子函数引用了外部父函数的变量或函数，当调用外部函数（即执行了内部函数定义），就产生了闭包（存在于内部子函数中）。

1.将内部函数作为外部函数的返回值。

```
function father() {
    var a = 1
    function son() {
        a++
        return a
    }
    return son
}
// 创建一个了内部函数对象
var f = father()
// 调用该内部函数对象，输出为2
console.log(f())
// 输出为3
console.log(f())
// 当包含闭包的函数对象成为垃圾对象时，闭包死亡。
f = null
```

局部变量`a`在函数调用后并未被销毁，延长了其生命周期。使得外部函数可以操纵内部数据。

2.将函数作为实参传递给另外一个函数。

```
function fun(message, time) {
    setTimeout(function () {
        // message导致了闭包
        console.log(message)
    }, time)
}
fun('测试', 1000)
```

如何利用闭包实现自定义`JS`模块?

`jQuery`就是通过该方法实现。

```
// 分号与形参的使用都是为了代码压缩
; (function (window) {
    var now
    function getStandardDate() {
        now = new Date()
        return now.getFullYear() + format(now.getMonth() + 1) +
format(now.getDate())
    }
    function format(num) {
        num = num.toString()
        return num.length == 1 ? '0' + num : num
    }
    window.DateUtil = {
```

```
        // 暴露的方法
        getStandardDate: getStandardDate
    }
})(window)
```

```
<head>
    <script type="text/javascript" src="./DateUtil.js"></script>
</head>
<script>
    console.log(DateUtil.getStandardDate())
</script>
```

什么是内存泄漏？常见的内存泄漏？

占用的内存未及时释放。

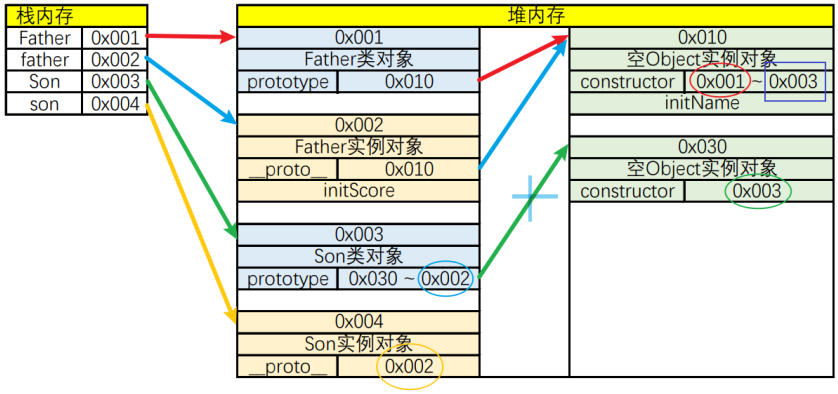
1.意外的全局变量。函数作用域中不使用`var`声明的变量将变成全局变量。

```
function fun() {
    c = 10
}
fun()
// 10
console.log(c)
```

2.闭包未死亡。

js如何实现继承？

```
// 1.
function Father(name, age) {
    this.name = name
    this.age = age
}
Father.prototype.initName = function () {
    this.name = 'default'
}
// 2.构造函数继承
function Son(name, age, score) {
    Father.call(this, name, age)
    this.score = score
}
// 3.原型链继承
Son.prototype = new Father()
// 4.校正显示原型的constructor属性指向
Son.prototype.constructor = Son
// 子类方法定义须在原型链继承之后
Son.prototype.initScore = function () {
    this.score = 0
}
// 5.使用
var son = new Son('songyx', 24, 90)
son.initName()
son.initScore()
// {name: 'default', age: 24, score: 0}
console.log(son)
```



什么是浏览器内核?

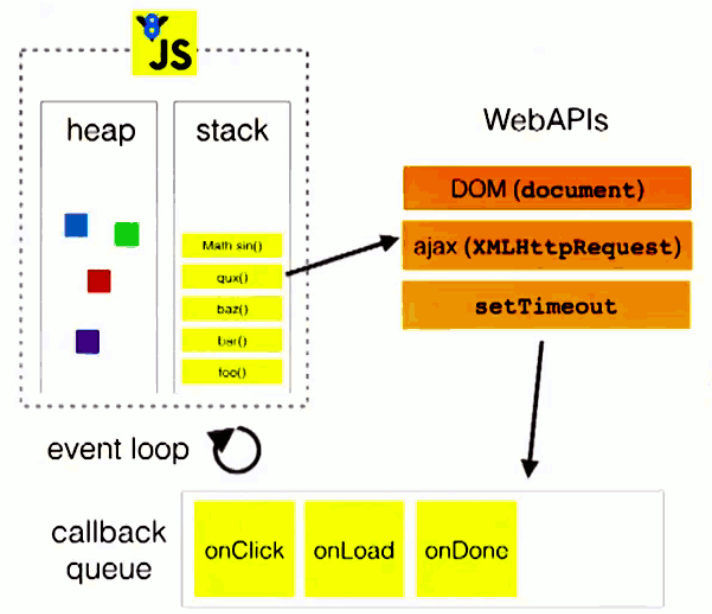
支撑浏览器运行的最核心的程序，由很多模块组成。

主线程：`js`引擎、`html,css`文档解析模块、`DOM/CSS`模块、布局和渲染模块。

分线程：定时器模块、事件响应模块、网络请求模块`Ajax`。

什么是事件循环模型?

`js`是单线程执行的。首先执行初始化代码（设置定时器、绑定监听、发送`Ajax`请求），并将其中的回调函数放入回调函数队列。之后遍历队列依次执行回调函数。



JS如何实现多线程?

使用`Web Workers`创建分线程。

```
<html>
<head>
    <script type="text/JavaScript" src="./worker.js"></script>
</head>
<body>
    <input class="number">
    <button id='btn'>点击</button>
</body>
</html>
<script>
    var input = document.querySelector('.number')
```

```
    // 创建worker对象
    var worker = new Worker('worker.js')
    document.getElementById('btn').onclick = function () {
        // 发送数据给分线程
        worker.postMessage(input.value)
        worker.onmessage = function (event) {
            alert(event.data)
        }
    }
</script>
```

使用`postMessage`双向通信。

```
// 分线程接收数据
var onmessage = function (event) {
    var number = event.data
    var result = calculate(number)
    // 发送数据回主线程
    postMessage(result)
    // 注意分线程中全局对象不再是window，alert()等方法失效
}
function calculate(number) {
    return number <= 2 ? 1 : calculate(number - 1) + calculate(number - 2)
}
```

不足：不能跨域加载`js`、`worker`内的代码不能访问DOM、有些浏览器不支持。