选择器

后代选择器

群选择器

```
p {
    color: rgb(123, 52, 133) !important;
  }
```

超链接伪类顺序

a:link：未访问的

visited：已经访问的

hover:位于链接上方

active：点击的时候

内容——内边距——边框——外边距

`display` 属性值的说明如下表所示。

属性值	说明
block	元素以块级方式展示。
inline	元素以内联方式展示。
inline-block	元素以内联块的方式展示。
none	隐藏元素。

css3新增选择器

- 在上面代码中，我们使用 `a[href^="#"]` 去匹配 `a` 标签中 `href` 属性值以 `#` 开头的元素。
- 使用 `a[href$="org"]` 去匹配 `a` 标签中 `href` 属性值以 `org` 结尾的元素。
- 使用 `a[href*="un"]` 去匹配 `a` 标签中 `href` 属性值包含 `un` 的元素。

伪类选择器

- 在代码中，我们使用 `div:nth-child(2)` 给 `div` 的第 2 个子元素添加绿色背景颜色。
- 使用 `div:nth-of-type(4)` 给父元素下第 4 个 `div` 子元素添加紫色背景颜色。

UI 伪类选择器是通过元素的状态来选择的一种选择器。

:focus	给获取焦点的元素设置样式。

input:focus { background-color: rgb(255, 153, 0); }

文本溢出设置

- 使用 `text-overflow: clip` 给 `class=poem1` 的 `p` 标签设置文本溢出时，修剪溢出的文本。
- 使用 `text-overflow: ellipsis` 给 `class=poem2` 的 `p` 标签设置用省略号代替溢出的文本。

元素转换

`transform` 是元素转换属性，其属性值为转换函数，使用该属性可以让元素向指定方向移动、缩放大小、旋转等变化。

在我们的实验中，会学到以下三种转换函数：

- 旋转函数（rotate）
- 移动函数（translate）
- 缩放函数（scale）

transform: rotate(角度); /元素按照指定角度旋转/

过渡

transition: 指定属性 持续时间 速度曲线 开始时间;

transition: transform 1s ease-in-out;

动画

`@keyframes` 被称为关键帧，它能够设置一些元素的样式，让该元素可以从原来的样式渐渐过渡到新的样式中。其语法格式如下所示：

```
@keyframes 动画名
{
    0% {样式属性：属性值;}
    25% {样式属性：属性值;}
    50% {样式属性：属性值;}
    100% {样式属性：属性值;}
}
```

这里的百分比是用来规定动画发生变化的时间的，`0%` 代表动画的开始，`100%` 代表动画的结束，中间的可以自定义。

将 `@keyframes` 创建的动画绑定到选择器上，通过 `animation` 属性就能实现动画效果了，其语法格式为：

```
animation: 动画名 完成动画的周期 是否重复;
```

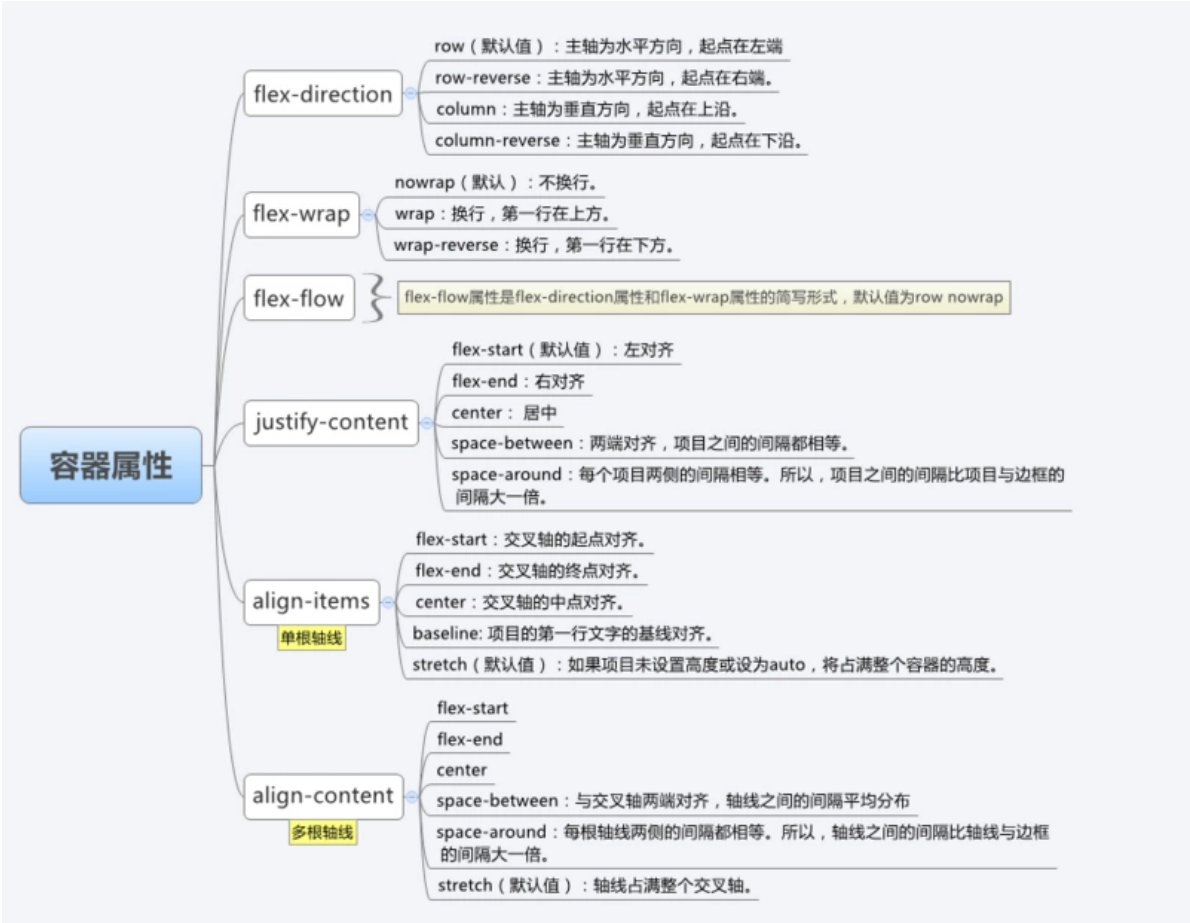
`animation` 属性是一个复合属性，它的子属性如下所示。

属性	描述
animation-name	规定 @keyframes 动画的名称。
animation-duration	规定动画完成一个周期所花费的秒或毫秒。默认是 0。

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .circle {
        width: 60px;
        height: 60px;
        border-radius: 100%;
        background-color: #ffd8a6;
        animation-name: action;
        animation-duration: 9s;
        animation-iteration-count: 10;
      }
      @keyframes action {
        0% {
          margin-left: 400px;
        }
        25% {
          background-color: #dd7631;
        }
        50% {
          border-radius: 10%;
        }
        100% {
          margin: 100px;
        }
      }
    </style>
  </head>
  <body>
    <div class="circle"></div>
  </body>
</html>
```

弹性布局

- `flex-direction` 属性，它指定了弹性子元素在父容器中的排列方向和顺序。
- `flex-wrap` 属性，它指定了弹性盒子的子元素换行方式。
- `align-items` 属性，它设置或检索弹性盒子元素在轴上对齐方式。
- `align-content` 属性，它是用于控制多行的对齐方式。



@media 媒体查询

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      @media screen and (max-width: 500px) {
        body {
          background-color: red;
```

```
        }
      }
      @media screen and (min-width: 800px) {
        body {
          background-color: green;
        }
      }
      @media screen and (min-width: 1024px) {
        body {
          background-color: blue;
        }
      }
    </style>
  </head>
  <body></body>
</html>
```

根据不同的页面，显示不同的样式

内置对象

`sort()` 可以给数组中的元素从小到大进行排序。

`join()` 可以将数组中的字符拼接成字符串。

`includes()` 可以用来判断该数组中是否包含某个元素。

`toString()` 可以将数组中的值转换成字符串类型。

`indexOf()` 可以用来查找指定元素的下标值。

`toLowerCase()` 可以把字符串的大写字母转换成小写字母。

toUpperCase()

`charAt()` 是用于根据指定下标从一个字符串中返回指定的字符。

str.substring(7, 10)); // 获取下标为 7-10 的字符

`split` 可以使用指定的分隔符将一个字符串分割成子字符串数组。

`indexOf()` 是寻找某个字符在字符串中首次出现的位置。

BOM的使用

window 对象的方法如下所示：

方法	描 述
alert()	显示一个警告框。
prompt()	显示可提示用户输入的对话框。
confirm()	显示一个有确认和取消按钮的对话框。
open()	打开一个新的浏览器窗口。
close()	关闭浏览器。
print()	打印当前窗口内容。

事件

常用的鼠标事件如下表所示：

事件	说明
onclick	鼠标点击事件
onmouseover	鼠标移入事件
onmouseout	鼠标移出事件
onmousedown	鼠标按下事件
onmouseup	鼠标松开事件
onmousemove	鼠标移动事件

常用的键盘事件只有以下两个：

- onkeydown：键盘按下会触发的事件。
- onkeyup：键盘松开会触发的事件。

在 JavaScript 中，常用表单事件如下表所示：

事件	说明
onfocus	表单元素聚焦时触发。
onblur	表单元素失焦时触发。

DOM 2 级事件的使用

DOM 2 级事件可以让多个事件执行。

在 DOM 2 级事件里，所有的 DOM 节点都有两个方法，分别是 addEvenetListener 和 removeEventListener。

```
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      body {
        text-align: center;
      }
    </style>
  </head>
  <body>
    <input id="btn" type="button" value="点我" />
    <script>
      document.getElementById("btn").onclick = function () {
        alert("1");
      };
      document.getElementById("btn").addEventListener("click", function () {
        alert("2");
      });
      document.getElementById("btn").addEventListener("click", function () {
        alert("3");
      });
    </script>
  </body>
</html>
```

事件对象的使用

value.onclick = function (e) { console.log("这是一个" + e.type + "事件"); // 控制台打印事件类型
};

原生 AJAX 技术

知识点

- XMLHttpRequest 对象
- open 和 send 方法
- onreadystatechange 函数

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>AJAX 的使用</title>
    <script>
      window.onload = function () {
        if (window.XMLHttpRequest) {
          // Mozilla, Safari, IE7+ 等浏览器适用
          var httpRequest = new XMLHttpRequest();
        } else if (window.ActiveXObject) {
          // IE 6 或者更老的浏览器适用
          var httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        }
```

```
        // 规定发送请求的一些要求
        httpRequest.open(
          "GET",
          "https://jsonplaceholder.typicode.com/users",
          true
        );
        // 将请求发送到服务器
        httpRequest.send();
        httpRequest.onreadystatechange = function () {
          console.log(httpRequest.readyState);
          console.log(httpRequest.status);
          if (httpRequest.readyState == 4 && httpRequest.status == 200) {
            // 请求成功执行的代码
            document.getElementById("item").innerHTML = "请求成功";
          } else {
            // 请求失败执行的代码
            document.getElementById("item").innerHTML = "请求失败";
          }
        };
      };
    </script>
  </head>
  <body>
    <div id="item"></div>
  </body>
</html>
```

正则表达式

常用表达式

表达式	描述
[a-z]	查找任何从小写 a 到小写 z 的字符
[A-Z]	查找任何从大写 A 到大写 Z 的字符
[0-9]	查找任何从 0 至 9 的数字
[abc]	查找括号内的任意一个字符
[^abc]	查找除了括号内的任意字符

常用的元字符（特殊字符）

字符	描述
\w	匹配数字、字母、下划线
\W	匹配非数字、字母、下划线
\d	匹配数字
\D	匹配非数字
\s	匹配空白字符（空格、换行）
\S	匹配非空白字符
\n	匹配换行符

常用的限定符

字符	描述
*	匹配前面的子表达式零次或多次
+	匹配前面的子表达式一次或多次
?	匹配前面的子表达式零次或一次
{n}	匹配确定的 n 次
{n,}	至少匹配 n 次
{n,m}	最少匹配 n 次且最多匹配 m 次

常用的修饰符

修饰符	描述
i	执行对大小写不敏感的匹配。
g	执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。
m	执行多行匹配。

其他

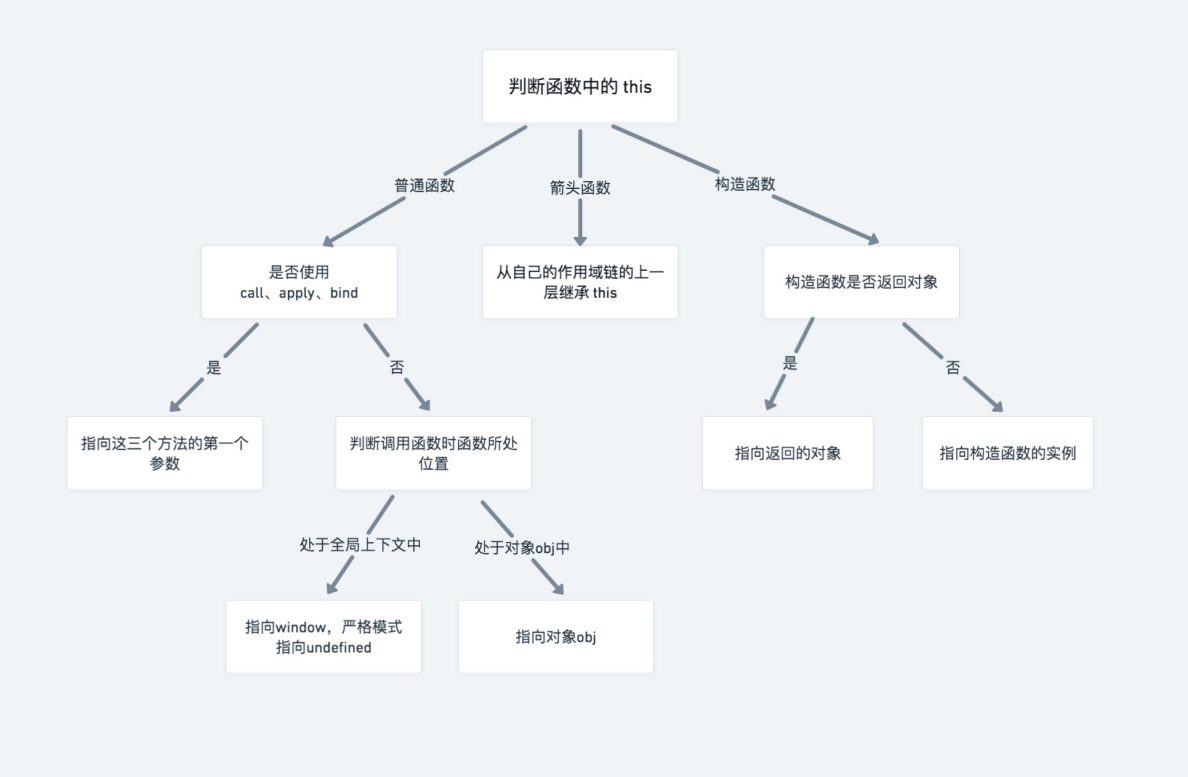
修饰符	描述
^	以...开始
$	以...结束

```
//验证规则：5-10个字符@1个或多个以上字符，以com、net、org结尾
        var regExp = /^\w{5,10}@\w+\.(com|net|org)$/;

        var flag = regExp.test(email);

        return flag;
```

将对象转化成数组，并判断长度

Object.keys(obj).length === 0;

this指向



剪头函数的this改变之后就不变了

分一分的代码

```
 * */
const splitArray = (oldArr, num) => {
  // TODO: 请补充代码实现功能
  let newArr = oldArr.sort((a,b)=>a-b)
  console.log(newArr);
  let arr = []
  for (let i = 0; i < newArr.length; i+=num) {
   arr.push(newArr.slice(i,num+i))
  }
  return arr
};
module.exports = splitArray; // 检测需要，请勿删除

// [4, 5, 6, 8, 10, 11, 12, 22, 23, 34, 76, 91]

// [4,5],[6,8],[10,11]
```

jQuery

jQuery 中的五种基本选择器

```
$(function () {
       $(".test").css({
         color: "blue",
         width: "200px",
         height: "80px",
         border: "2px solid yellow",
       });
     });
```

- 后代选择器$("M N");选择所有后代
- 子代选择器$("M>N");就只选择儿子
- 兄弟选择器$("M~N");
- 相邻选择器$("M+N"); // 选择下一个兄弟节点 N

属性选择器

- 在上面代码中，`div[id]` 选择了带有 ID 选择器的元素。
- `div[id='item2']` 选择了 id 名为 item2 的元素。
- `div[id !='item2']` 选择了 id 名不为 item2 的元素。
- `body[class ^='c']` 选择了 class 名以 c 开头的元素。

伪类选择器

- 在上面代码中，使用 `$("td:contains('1')")` 给 `td` 元素中包含内容 1 的元素设置绿色字体。
- 使用 `$("tr:has(td)")` 给包含 `td` 元素的 `tr` 元素设置背景颜色。
- 使用 `$("td:empty")` 给空的 `td` 元素添加文字内容。
- 使用 `$("td:parent")` 给是父亲的 `td` 元素添加文字内容。
- 在上面代码中，使用 `$("input:text")` 给 text 类型的表单元素设置背景颜色为黄色。

DOM操作

本节实验给大家介绍了几种常用的 DOM 操作。分别是：

- 节点的创建：使用 `$()` 来创建元素节点。
- 元素的插入：分为子级插入和同级插入。
  - 子级插入方法，包括 `prepend()`、`prependTo()`、`append()`、`appendTo()`。
  - ```
    // 在 A 元素的子级最前面的位置插入B
    $(A).prepend(B);
    ```
  - 同级插入方法，包括 `before()`、`insertBefore()`、`after()`、`insertAfter()`。
- 元素的删除：使用 `remove()` 或者 `empty()` 方法可以删除元素。

  $("div").remove();
- 元素的替换：使用 `replaceWith()` 或者 `replaceAll()` 可以替换元素。
  - // 将 A 替换为 B　$(A).replaceWith(B);
    ```

- 元素的遍历：使用 `each` 可以遍历元素。
- 属性操作：使用 `attr()` 设置属性、获取属性和使用 `removeAttr` 删除属性。
  - // 2.使用 attr 方法来修改 src 的属性值　　$("img").attr("src", "bird2.jpg");
- 样式操作：使用 `css()` 获取样式，使用 `addClass()` 添加样式，使用 `removeClass()` 移除样式，使用 `toggleClass()` 切换类选择器。
  - $("div").css("border", "4px solid #94ebcd"); // 给 div 元素添加一个边框
- 内容操作：使用 `html()` 获取指定元素内部的元素标签和标签中的内容，使用 `text()` 获取指定标签里的文本或者给指定标签添加文本，使用 `val()` 获取表单元素的值或者给表单元素设置值。
  - // 获取元素内容 $().html();
  - // 设置元素内容 $().html("内容");

动画

- 隐藏（hide）与显示（show）
  - $().show(speed, easing, callback);
  - ```
    $("div")
              .mouseover(function () {
                $(this).hide(); // 鼠标移入时，隐藏该元素
              })
    ```

- 淡入（fadeIn）与淡出（fadeOut）：同上
- 自定义动画（animate）
- 队列动画（多个 animate）
  - ```
              // 改变标题中字的间距和字的大小
              $("#title")
                .animate({ letterSpacing: "5px" })
                .animate({ fontSize: "25px" });
    ```

- 回调函数（动画执行完毕以后的回调）
- 停止动画（stop）
  - $().stop(stopAll, goToEnd); 停止所有动画，跳转到最后
- 延迟动画（delay）
  - $("div").delay(3000).animate({ "background-color": "#ddffbc" });

遍历元素的方法

- 遍历祖先元素：parent 和 parents 方法可以用来查找指定元素的祖先元素。
  - ```
    $("div").hover(
              function () {
                // 当鼠标放在元素上时，给整个列表添加一个背景颜色
                $("li").parent().css("background-color", "#edffec");
              },
    ```
    ```

- 遍历兄弟元素：有三种兄弟元素查找的方法，分别为
  - 前向兄弟元素查找 prev、preAll。
    - $("#div3").prev().css("background", "#a6d6d6");
  - 后向兄弟元素查找 next、nextAll。
  - 所有兄弟元素查找 siblings()。
- 遍历后代元素：children 和 find。
  - 子代、所有后代
- 过滤元素：有四种过滤方法，分别为
  - 类名过滤 hasClass。
  - 下标过滤 eq。取值为元素的下标值
  - 判断过滤 is。
  - 反向过滤 not。不符合条件的情况

去重

return [...new Set(arr)];

// 遍历：遇到{的情况，就把元素存入数组，然后转化成字符串，然后提取出来

// 最后匹配到data的值，然后替换

实现模板字符串解析——利用正则

```
function strRender(str, data) {
  // 正则中加入全局搜索修饰符，也就是在尾部加一个 g
  const re = /\${(\w+)\}/g;
  return str.replace(re, (match, key) => data[key]);
}
```

这里 replace 函数的替换值就不再是一个字符串了，而是一个回调函数，这个回调函数每次匹配都会调用，每次调用时会把匹配到了的模板变量替换为真实的数据，这样就不需要递归调用了。用到的原理就是 replace 函数可以指定一个函数作为参数，