

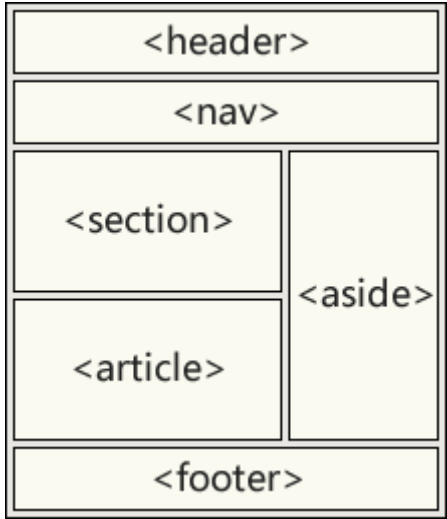
HTML

一、语义标签

html5语义标签，可以使开发者更方便清晰构建页面的布局

标签	描述
	定义了文档的头部区域
	定义了文档的尾部区域
	定义文档的导航
	定义文档中的节
	定义文章
	定义页面以外的内容
► 详细信息	定义用户可以看到或者隐藏的额外细节
	标签包含details元素的标题

标签	描述
	定义对话框
	定义自包含内容，如图表
	定义文档主内容
	定义文档的主内容
	定义日期/时间




二、增强型表单

html5修改一些新的input输入特性，改善更好的输入控制和验证

输入类型	描述
------	----

输入类型	描述
color	主要用于选取颜色
date	选取日期
datetime	选取日期(UTC时间)
datetime-local	选取日期（无时区）
month	选择一个月份
week	选择周和年
time	选择一个时间
email	包含e-mail地址的输入域
number	数值的输入域
url	url地址的输入域
tel	定义输入电话号码和字段
search	用于搜索域
range	一个范围内数字值的输入域

html5新增了五个表单元素

用户会在他们输入数据时看到域定义选项的下拉列表	
	进度条，展示连接/下载进度
	刻度值，用于某些计量，例如温度、重量等
	提供一种验证用户的可靠方法生成一个公钥和私钥
	用于不同类型的输出比如尖酸或脚本输出

html5新增表单属性

属性	描述
placeholder	输入框默认提示文字
required	要求输入的内容是否可为空
pattern	描述一个正则表达式验证输入的值
min/max	设置元素最小/最大值
step	为输入域规定合法的数字间隔
height/wdith	用于image类型  标签图像高度/宽度
autofocus	规定在页面加载时，域自动获得焦点
multiple	规定  元素中可选择多个值

三、音频和视频

html5提供了音频和视频文件的标准，既使用元素。

音频：

```
<audio controls>      //controls属性提供添加播放、暂停和音量控件。
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
  您的浏览器不支持 audio 元素。      //浏览器不支持时显示文字
</audio>
```

视频：

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  您的浏览器不支持Video标签。
</video>
```

四、CANVAS绘图

`<canvas>` 是 HTML5 新增的，一个可以使用脚本(通常为 JavaScript) 在其中绘制图像的 HTML 元素。它可以用来制作照片集或者制作简单(也不是那么简单)的动画，甚至可以进行实时视频处理和渲染。



```
<canvas id="tutorial" width="300" height="300"></canvas>
```

属性只有 width 和 height 两个属性，默认为 300px、150px。

注：不要使用 css 来改变 canvas 宽高、会导致比例不一致出现扭曲。

有些老的浏览器不支持 canvas 元素，可以这样替换：

用文本替换：



```
<canvas>
  你的浏览器不支持 canvas，请升级你的浏览器。
</canvas>
```

用 `` 替换：



```
<canvas>
  
</canvas>
```

结束标签 `</canvas>` 不可省略。

H3 绘制矩形

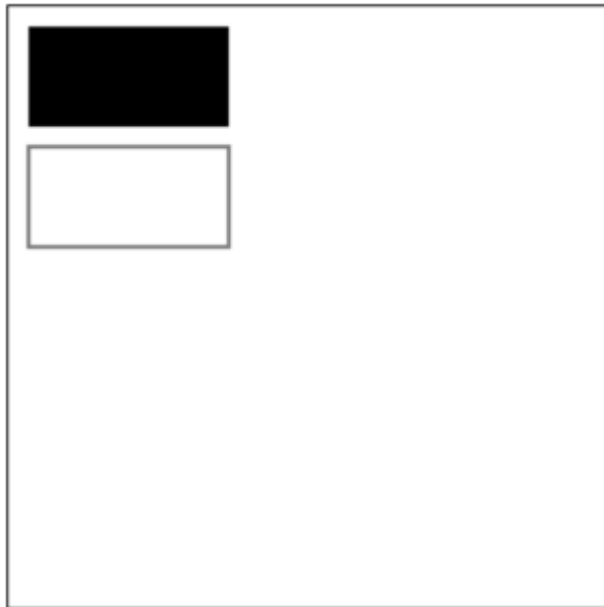
canvast 提供了三种方法绘制矩形：

- 1、**fillRect(x, y, width, height)**：绘制一个填充的矩形。
- 2、**strokeRect(x, y, width, height)**：绘制一个矩形的边框。
- 3、**clearRect(x, y, width, height)**：清除指定的矩形区域，然后这块区域会变的完全透明。

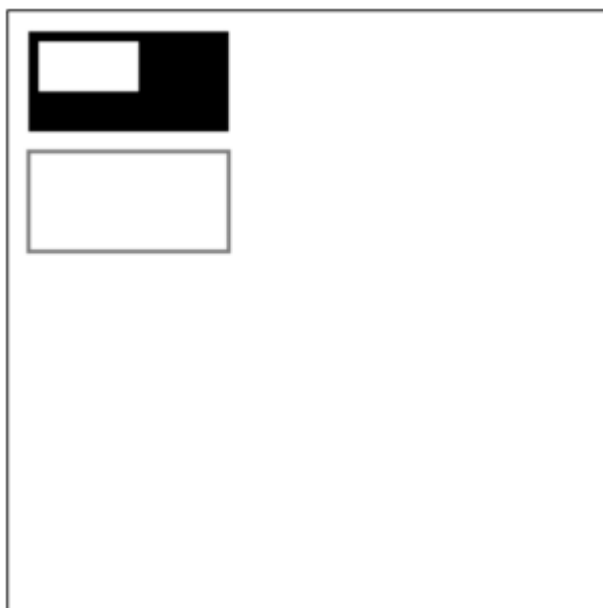
说明：这 3 个方法具有相同的参数。

- **x, y** : 指的是矩形的左上角的坐标。(相对于canvas的坐标原点)
- **width, height** : 指的是绘制的矩形的宽和高。

```
function draw(){  var canvas = document.getElementById('tutorial');  if(!canvas.getContext)
return;  var ctx = canvas.getContext("2d");  ctx.fillRect(10, 10, 100, 50);  // 绘制矩形, 填充的默认颜色为黑色  ctx.strokeRect(10, 70, 100, 50); // 绘制矩形边框  } draw();
```



```
ctx.clearRect(15, 15, 50, 25);
```



H3 绘制路径

方法:

01. `beginPath()`

新建一条路径，路径一旦创建成功，图形绘制命令被指向到路径上生成路径

02. `moveTo(x, y)`

把画笔移动到指定的坐标 `(x, y)`。相当于设置路径的起始点坐标。

03. `closePath()`

闭合路径之后，图形绘制命令又重新指向到上下文中

04. `stroke()`

通过线条来绘制图形轮廓

05. `fill()`

通过填充路径的内容区域生成实心的图形

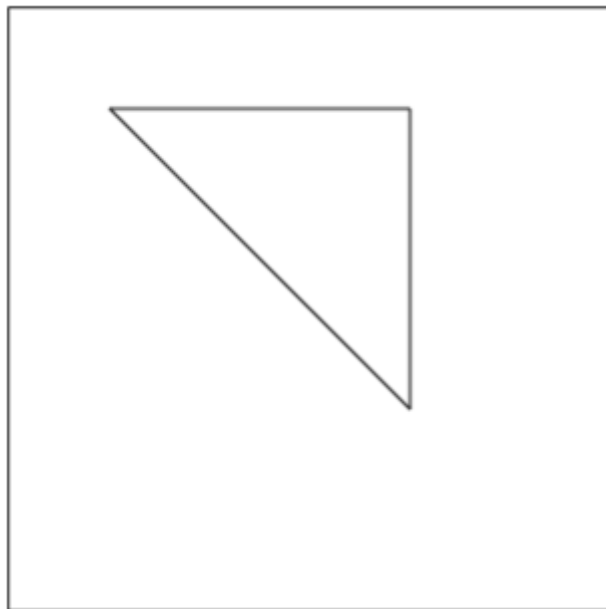
绘制线段

```
function draw(){
    var canvas = document.getElementById('tutorial');
    if (!canvas.getContext) return;
    var ctx = canvas.getContext("2d");
    ctx.beginPath(); //新建一条path
    ctx.moveTo(50, 50); //把画笔移动到指定的坐标
    ctx.lineTo(200, 50); //绘制一条从当前位置到指定坐标(200, 50)的直线.
    //闭合路径。会拉一条从当前点到path起始点的直线。如果当前点与起始点重合，则什么
    都不做
    ctx.closePath();
    ctx.stroke(); //绘制路径。
}
draw();
```

绘制三角形边框

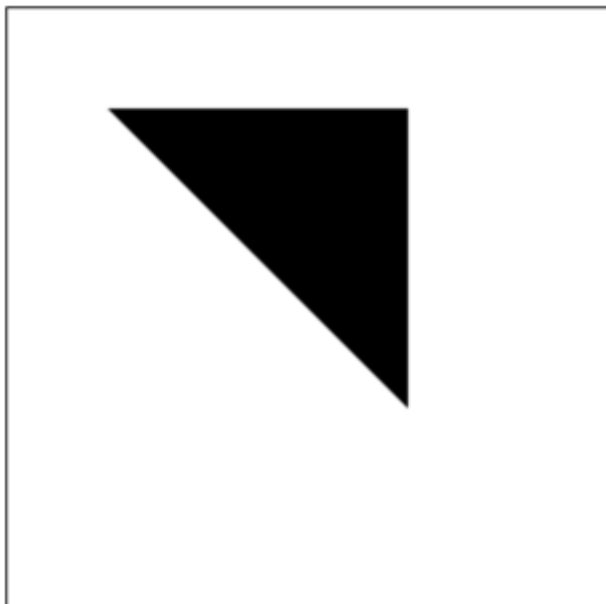


```
function draw(){
    var canvas = document.getElementById('tutorial');
    if (!canvas.getContext) return;
    var ctx = canvas.getContext("2d");
    ctx.beginPath();
    ctx.moveTo(50, 50);
    ctx.lineTo(200, 50);
    ctx.lineTo(200, 200);
    ctx.closePath(); //虽然我们只绘制了两条线段，但是closePath会closePath，
    仍然是一个3角形
    ctx.stroke(); //描边。stroke不会自动closePath()
}
draw();
```



填充三角形

```
function draw(){
    var canvas = document.getElementById('tutorial');
    if (!canvas.getContext) return;
    var ctx = canvas.getContext("2d");
    ctx.beginPath();
    ctx.moveTo(50, 50);
    ctx.lineTo(200, 50);
    ctx.lineTo(200, 200);
    ctx.fill(); //填充闭合区域。如果path没有闭合，则fill()会自动闭合路径。
}
draw();
```



H3 案例

01. 太阳系:

02.

```
let sun;
let earth;
let moon;
let ctx;
function init(){
    sun = new Image();
    earth = new Image();
    moon = new Image();
    sun.src = "sun.png";
    earth.src = "earth.png";
    moon.src = "moon.png";

    let canvas = document.querySelector("#solar");
    ctx = canvas.getContext("2d");

    sun.onload = function (){
        draw()
    }
}
init();
function draw(){
    ctx.clearRect(0, 0, 300, 300); //清空所有的内容
```

```

    /*绘制 太阳*/
    ctx.drawImage(sun, 0, 0, 300, 300);

    ctx.save();
    ctx.translate(150, 150);

    //绘制earth轨道
    ctx.beginPath();
    ctx.strokeStyle = "rgba(255,255,0,0.5)";
    ctx.arc(0, 0, 100, 0, 2 * Math.PI)
    ctx.stroke()

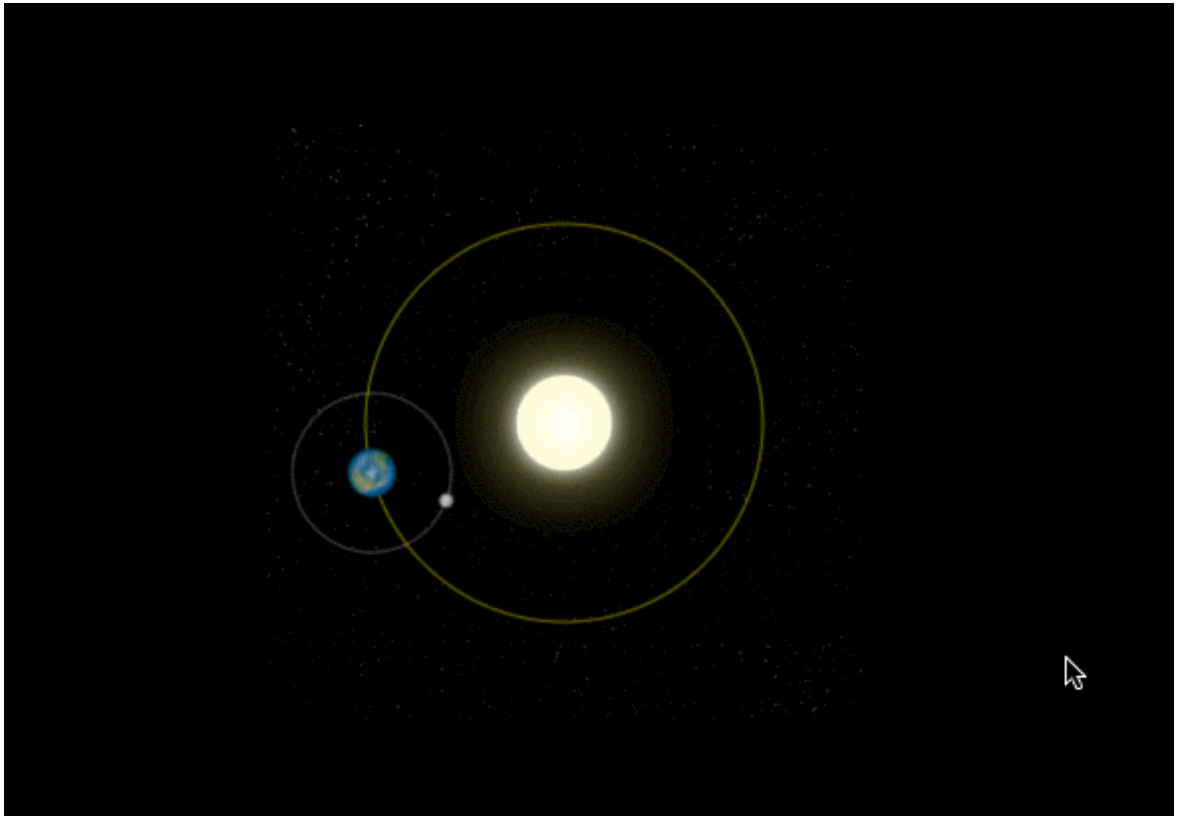
    let time = new Date();
    //绘制地球
    ctx.rotate(2 * Math.PI / 60 * time.getSeconds() + 2 * Math.PI /
60000 * time.getMilliseconds());
    ctx.translate(100, 0);
    ctx.drawImage(earth, -12, -12)

    //绘制月球轨道
    ctx.beginPath();
    ctx.strokeStyle = "rgba(255,255,255,.3)";
    ctx.arc(0, 0, 40, 0, 2 * Math.PI);
    ctx.stroke();

    //绘制月球
    ctx.rotate(2 * Math.PI / 6 * time.getSeconds() + 2 * Math.PI /
6000 * time.getMilliseconds());
    ctx.translate(40, 0);
    ctx.drawImage(moon, -3.5, -3.5);
    ctx.restore();

    requestAnimationFrame(draw);
}

```



地理定位

使用getCurrentPosition()方法来获取用户的位置。以实现“LBS服务”

```
<script>
    var x=document.getElementById("demo");
    function getLocation()
    {
        if (navigator.geolocation)
        {
            navigator.geolocation.getCurrentPosition(showPosition);
        }
        else{x.innerHTML="Geolocation is not supported by this browser.";}
    }
    function showPosition(position)
    {
        x.innerHTML="Latitude: " + position.coords.latitude +
            "<br />Longitude: " + position.coords.longitude;
    }
</script>
```

拖放API

拖放是一种常见的特性，即提取对象以后拖到另一个位置。

在html5中，拖放是标准的一部分，任何元素都能够拖放。



```
<div draggable="true"></div>
```

当元素拖动时，我们可以检查其拖动的数据。



```
<div draggable="true" ondragstart="drag(event)"></div>
<script>
function drap(ev){
    console.log(ev);
}
</script>
```

拖动生命周期	属性名	描述
拖动开始	ondragstart	在拖动操作开始时执行脚本
拖动过程中	ondrag	只要脚本在被拖动就运行脚本
拖动过程中	ondragenter	当元素被拖动到一个合法的防止目标时，执行脚本
拖动过程中	ondragover	只要元素正在合法的防止目标上拖动时，就执行脚本

拖动生命周期	属性名	描述
拖动过程中	ondragleave	当元素离开合法的防止目标时
拖动结束	ondrop	将被拖动元素放在目标元素内时运行脚本
拖动结束	ondragend	在拖动操作结束时运行脚本

FLEX布局

一、FLEX 布局是什么？

Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

任何一个容器都可以指定为 Flex 布局。



```
.box{  
display: flex;  
}
```

行内元素也可以使用 Flex 布局。



```
.box{  
display: inline-flex;  
}
```

Webkit 内核的浏览器，必须加上 `-webkit` 前缀。

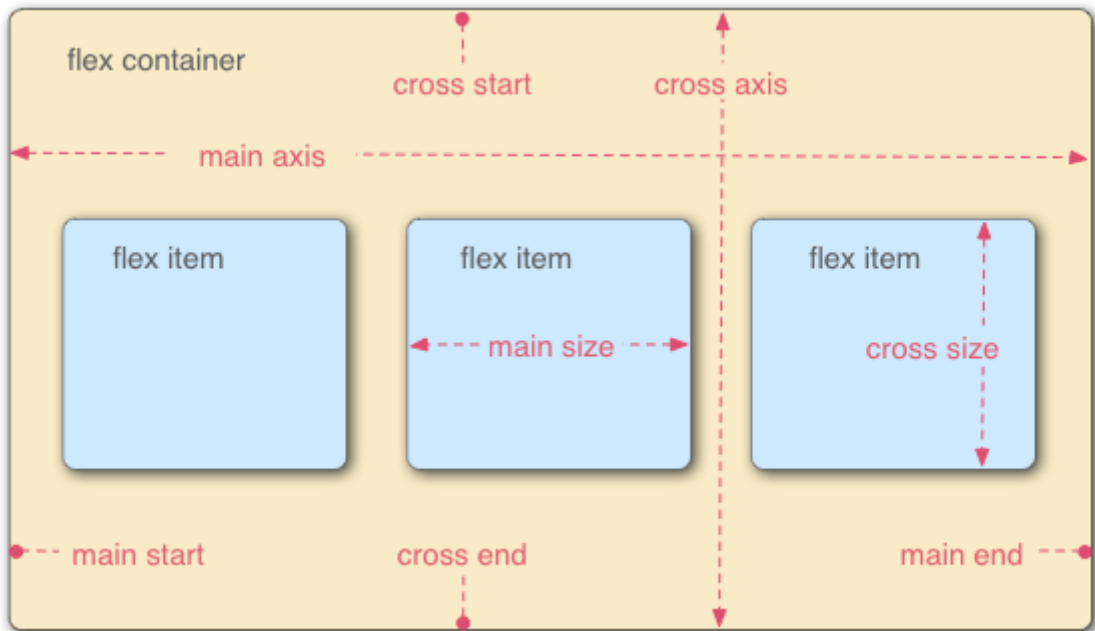


```
.box{  
display: -webkit-flex; /* Safari */  
display: flex;  
}
```

注意，设为 Flex 布局以后，子元素的 `float`、`clear` 和 `vertical-align` 属性将失效。

二、基本概念

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称“容器”。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称“项目”。



容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做 `main start`，结束位置叫做 `main end`；交叉轴的开始位置叫做 `cross start`，结束位置叫做 `cross end`。

项目默认沿主轴排列。单个项目占据的主轴空间叫做 `main size`，占据的交叉轴空间叫做 `cross size`。

三、容器的属性

以下6个属性设置在容器上。

- `flex-direction`
- `flex-wrap`
- `flex-flow`
- `justify-content`
- `align-items`
- `align-content`

H3 flex-direction属性

`flex-direction` 属性决定主轴的方向（即项目的排列方向）。



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

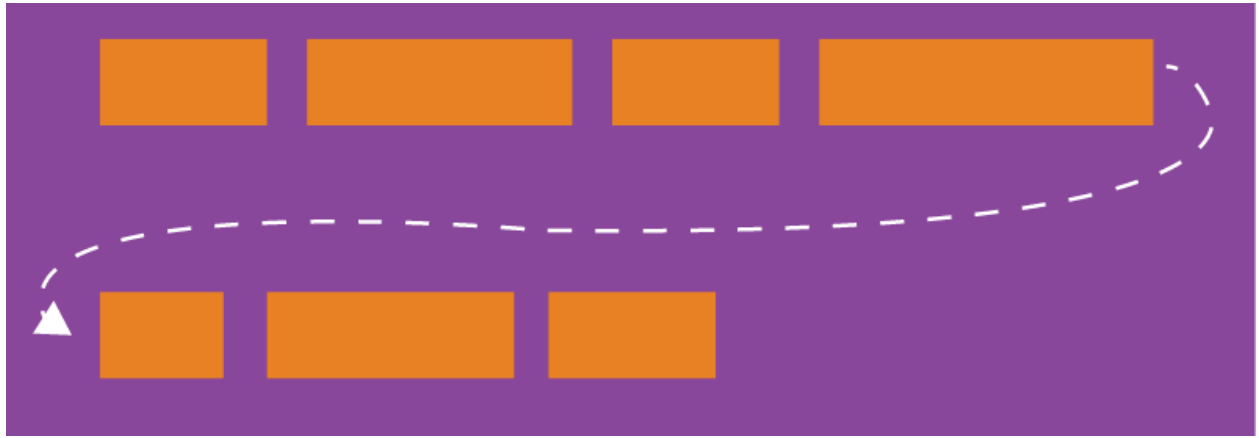


它可能有4个值。

- `row`（默认值）： 主轴为水平方向，起点在左端。
- `row-reverse`： 主轴为水平方向，起点在右端。
- `column`： 主轴为垂直方向，起点在上沿。
- `column-reverse`： 主轴为垂直方向，起点在下沿。

H3 flex-wrap属性

默认情况下，项目都排在一条线（又称“轴线”）上。`flex-wrap` 属性定义，如果一条轴线排不下，如何换行。



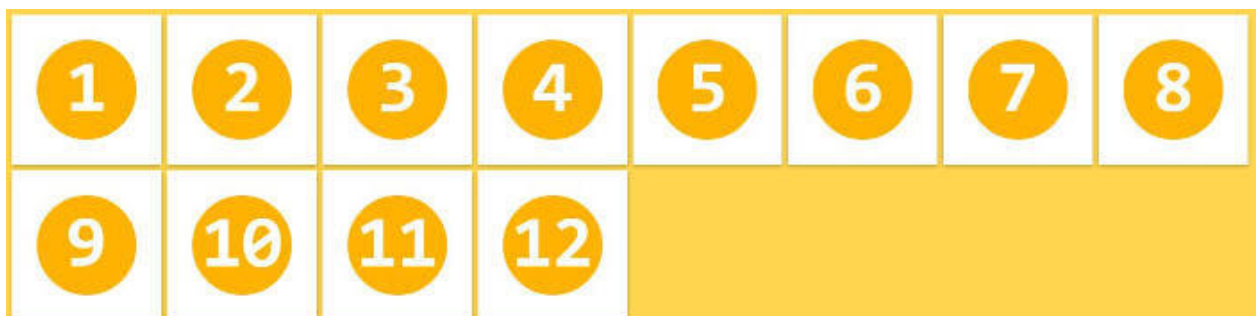
```
.box{  
flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

它可能取三个值。

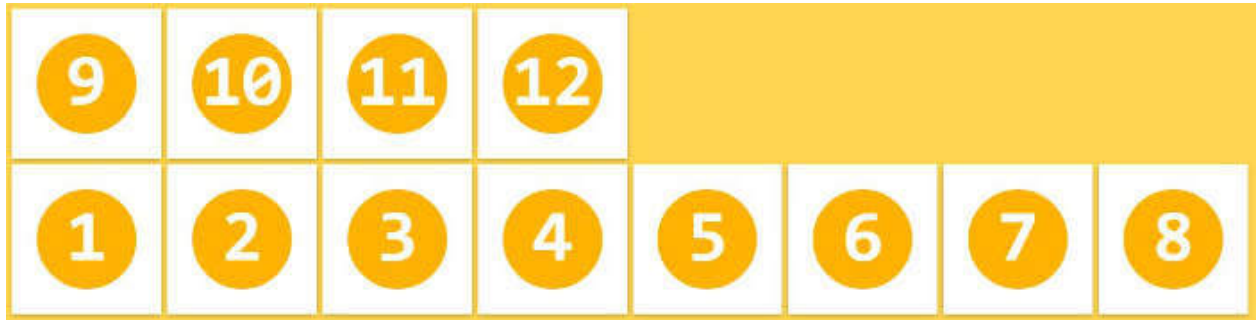
(1) `nowrap` (默认)：不换行。



(2) `wrap`：换行，第一行在上方。



(3) `wrap-reverse`：换行，第一行在下方。



H3 flex-flow

`flex-flow` 属性是 `flex-direction` 属性和 `flex-wrap` 属性的简写形式，默认值为 `row nowrap`。



```
.box {  
  flex-flow: <flex-direction> || <flex-wrap>;  
}
```

H3 justify-content属性

`justify-content` 属性定义了项目在主轴上的对齐方式。



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

flex-start



flex-end



center



space-between



space-around



它可能取5个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

- `flex-start`（默认值）：左对齐
- `flex-end`：右对齐
- `center`：居中
- `space-between`：两端对齐，项目之间的间隔都相等。

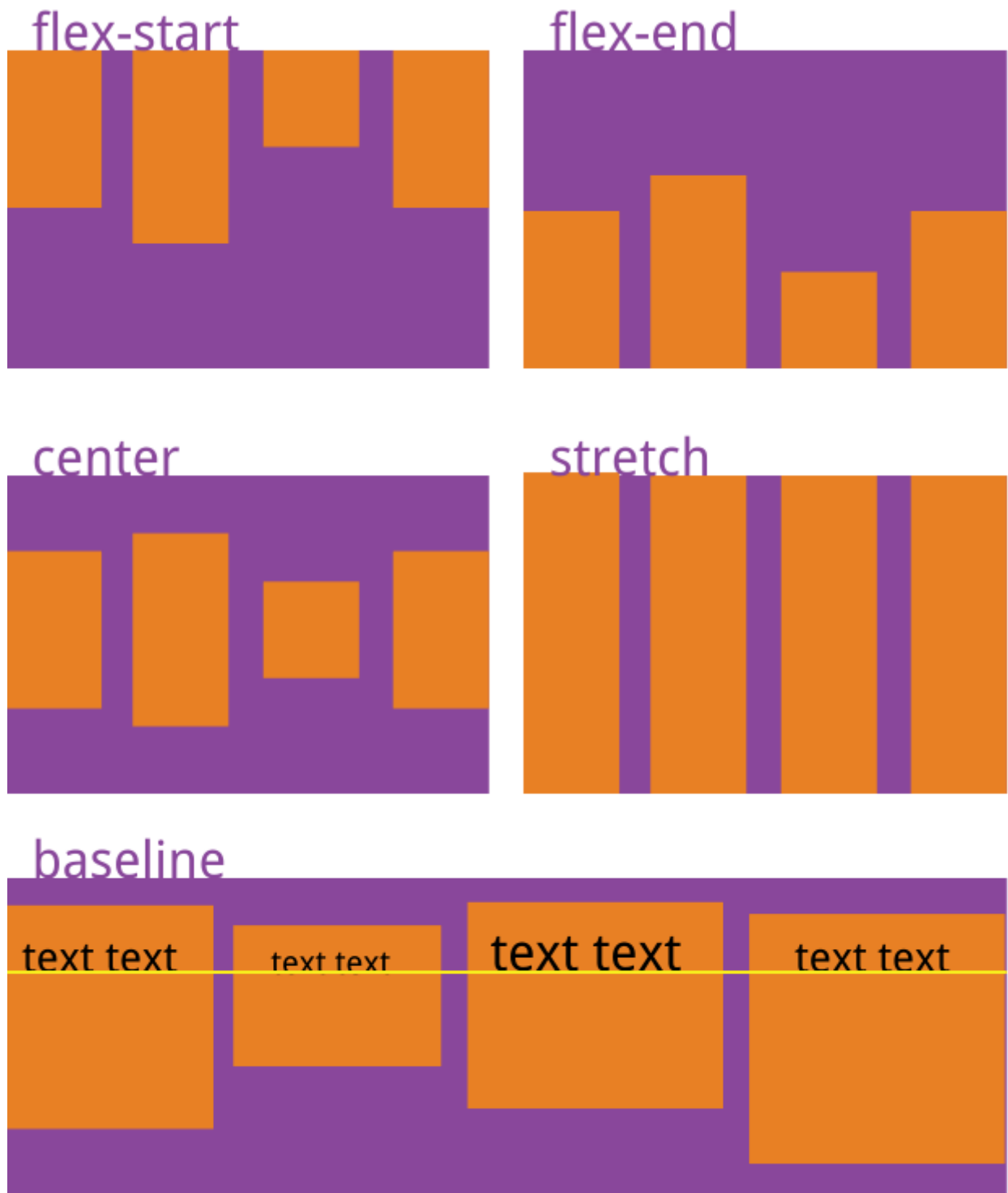
- **space-around**：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

H3 align-items属性

align-items 属性定义项目在交叉轴上如何对齐。



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



它可能取5个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

- **flex-start**：交叉轴的起点对齐。
- **flex-end**：交叉轴的终点对齐。
- **center**：交叉轴的中点对齐。
- **baseline**：项目的第一行文字的基线对齐。

- **stretch**（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

H3 align-content属性

align-content 属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。



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

flex-start



flex-end



center



stretch



space-between



space-around



该属性可能取6个值。

- **flex-start**：与交叉轴的起点对齐。
- **flex-end**：与交叉轴的终点对齐。
- **center**：与交叉轴的中点对齐。
- **space-between**：与交叉轴两端对齐，轴线之间的间隔平均分布。

- `space-around`：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
- `stretch`（默认值）：轴线占满整个交叉轴。

H3 项目的属性

以下6个属性设置在项目上。

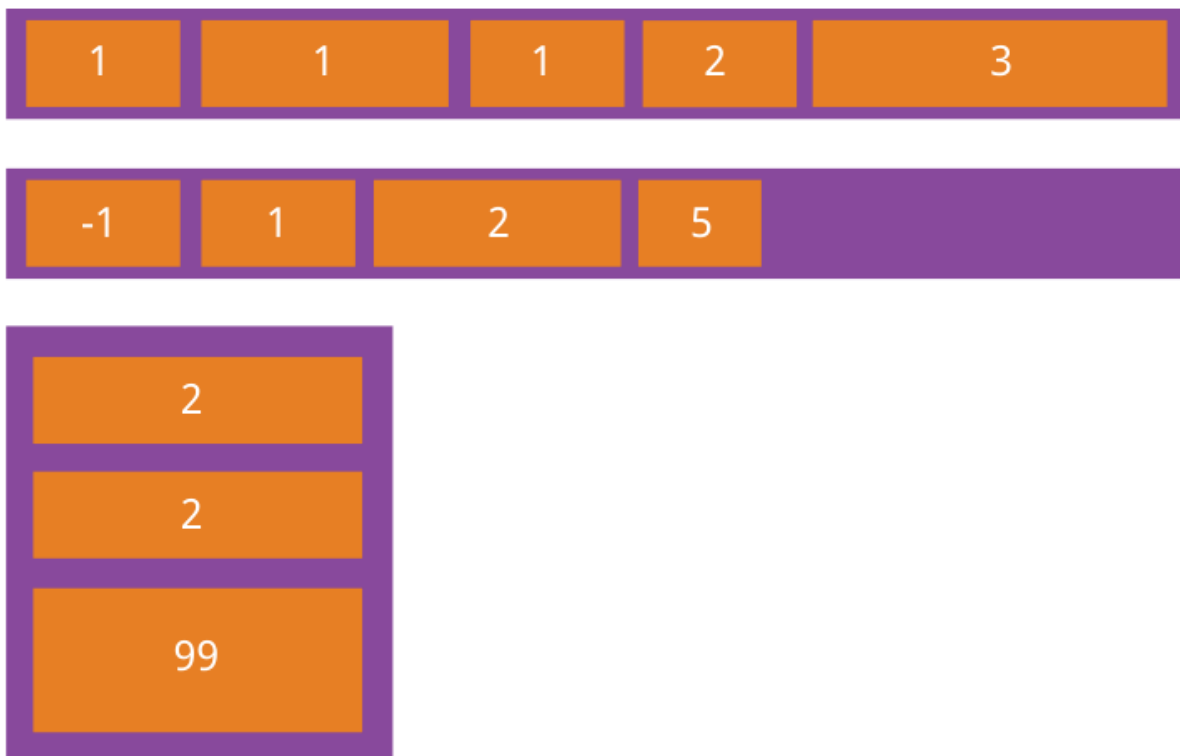
- `order`
- `flex-grow`
- `flex-shrink`
- `flex-basis`
- `flex`
- `align-self`

H3 order属性

`order` 属性定义项目的排列顺序。数值越小，排列越靠前，默认为0。



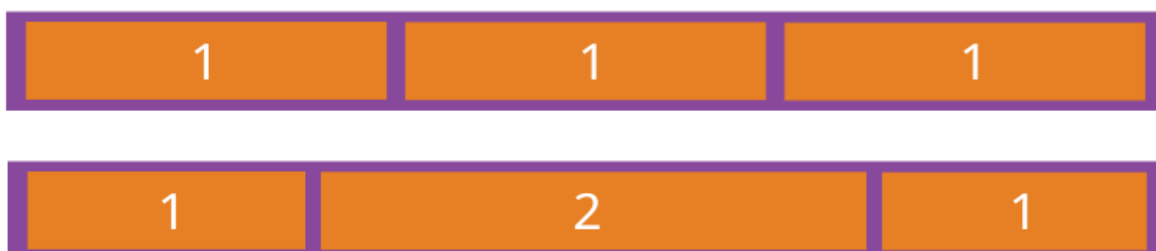
```
.item {  
  order: <integer>;  
}
```



H3 flex-grow属性

`flex-grow` 属性定义项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。

```
.item {  
  flex-grow: <number>; /* default 0 */  
}
```

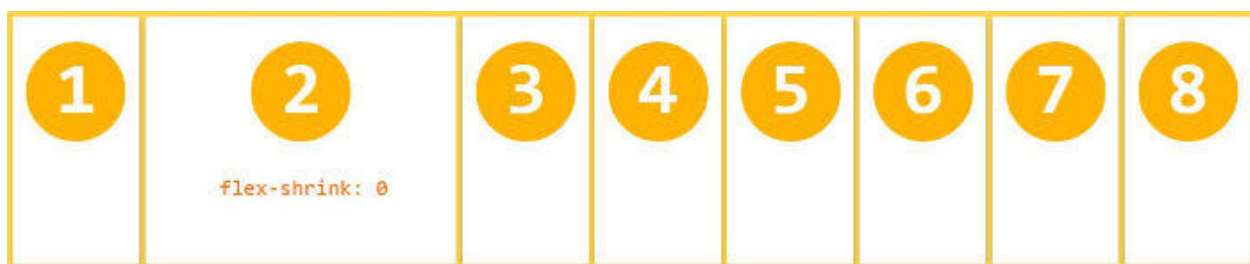


如果所有项目的 `flex-grow` 属性都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的 `flex-grow` 属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍。

H3 flex-shrink属性

`flex-shrink` 属性定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。

```
.item {  
  flex-shrink: <number>; /* default 1 */  
}
```



如果所有项目的 `flex-shrink` 属性都为1，当空间不足时，都将等比例缩小。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。

负值对该属性无效。

H3 flex-basis属性

`flex-basis` 属性定义了再分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为 `auto`，即项目的本来大小。



```
.item {  
  flex-basis: <length> | auto; /* default auto */  
}
```

它可以设为跟 `width` 或 `height` 属性一样的值（比如350px），则项目将占据固定空间。

H3 flex属性

`flex` 属性是 `flex-grow`，`flex-shrink` 和 `flex-basis` 的简写，默认值为 `0 1 auto`。后两个属性可选。



```
.item {  
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]  
}
```

该属性有两个快捷值：`auto` (`1 1 auto`) 和 `none` (`0 0 auto`)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

H3 align-self属性

`align-self` 属性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` 属性。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;  
}
```

flex-start



flex-end

该属性可能取6个值，除了auto，其他都与align-items属性完全一致。

AJAX

创建 XMLHTTPRequest



```
if(window.XMLHttpRequest){  
    // IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码  
    var xmlhttp = new XMLHttpRequest()  
}else{  
    // IE6, IE5 浏览器执行代码  
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP")  
}
```

发送请求：

GET请求：



```
xmlhttp.open("GET", "ajax_info.txt", true)  
xmlhttp.send();
```

方法	描述
<code>open(method , url , async)</code>	规定请求的类型、URL 以及是否异步处理请求。 <code>method</code> ：请求的类型；GET 或 POST <code>url</code> ：文件在服务器上的位置 <code>async</code> ：true（异步）或 false（同步）
<code>send(string)</code>	将请求发送到服务器。 <code>string</code> ：仅用于 POST 请求

在上面的例子中，您可能得到的是缓存的结果。

为了避免这种情况，请向 URL 添加一个唯一的 ID：

实例



```
xmlhttp.open("GET", "/try/ajax/demo_get.php?t=" + Math.random(), true);  
xmlhttp.send();
```

POST请求:



```
xmlhttp.open("POST", "/try/ajax/demo_post.php", true)  
xmlhttp.send();
```

POST接受HTML表单:



```
xmlhttp.open("POST", "/try/ajax/demo_post2.php", true);  
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-  
urlencoded");  
xmlhttp.send("fname=Henry&lname=Ford");  
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

服务器响应

responseText 属性

如果来自服务器的响应并非 XML，请使用 responseText 属性。

responseText 属性返回字符串形式的响应，因此您可以这样使用：



```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

responseXML 属性

如果来自服务器的响应是 XML，而且需要作为 XML 对象进行解析，请使用 responseXML 属性：

请求 cd_catalog.xml 文件，并解析响应：

```
xmlDoc=xmlhttp.responseXML;
txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)
{
    txt=txt + x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("myDiv").innerHTML=txt;
```

实例

异步加载 JSON 数据

```
function loadXMLDoc()
{
    var xmlhttp;
    if (window.XMLHttpRequest)
    {
        // IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
        xmlhttp=new XMLHttpRequest();
    }
    else
    {
        // IE6, IE5 浏览器执行代码
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()
    {
```



```

    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
        var myArr = JSON.parse(this.responseText);
        myFunction(myArr)
    }
}
xmlhttp.open("GET", "/try/ajax/json_ajax.json", true);
xmlhttp.setRequestHeader("Content-Type",
"application/json; charset=UTF-8");
xmlhttp.send();
}
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '"' +
        arr[i].title + '</a><br>';
    }
    document.getElementById("myDiv").innerHTML=out;
}

```

json_ajax.js



```

[
  {
    "title": "JavaScript 教程",
    "url": "https://www.runoob.com/js/"
  },
  {
    "title": "HTML 教程",
    "url": "https://www.runoob.com/html/"
  },
  {
    "title": "CSS 教程",
    "url": "https://www.runoob.com/css/"
  }
]

```