```
div2.setAttribute("class", "box");

console.log(div1.isSameNode(div1));   // true
console.log(div1.isEqualNode(div2));  // true
console.log(div1.isSameNode(div2));   // false
```

这里创建了包含相同属性的两个 元素。这两个元素相等，但不相同。

DOM3 也增加了给 DOM 节点附加额外数据的方法。setUserData()方法接收 3 个参数：键、值、处理函数，用于给节点追加数据。可以像下面这样把数据添加到一个节点：

```
document.body.setUserData("name", "Nicholas", function() {});
```

然后，可以通过相同的键再取得这个信息，比如：

```
let value = document.body.getUserData("name");
```

setUserData()的处理函数会在包含数据的节点被复制、删除、重命名或导入其他文档的时候执行，可以在这时候决定如何处理用户数据。处理函数接收 5 个参数：表示操作类型的数值（1 代表复制，2 代表导入，3 代表删除，4 代表重命名）、数据的键、数据的值、源节点和目标节点。删除节点时，源节点为 null；除复制外，目标节点都为 null。

```
let div = document.createElement("div");
div.setUserData("name", "Nicholas", function(operation, key, value, src, dest) {
  if (operation == 1) {
    dest.setUserData(key, value, function() {});  }
});

let newDiv = div.cloneNode(true);
console.log(newDiv.getUserData("name"));  // "Nicholas"
```

这里先创建了一个 元素，然后给它添加了一些数据，包含用户的名字。在使用 cloneNode()复制这个元素时，就会调用处理函数，从而将同样的数据再附加给复制得到的目标节点。然后，在副本节点上调用 getUserData()能够取得附加到源节点上的数据。

4. 内嵌窗格的变化

DOM2 HTML 给 HTMLIFrameElement（即<iframe>，内嵌窗格）类型新增了一个属性，叫 contentDocument。这个属性包含代表子内嵌窗格中内容的 document 对象的指针。下面的例子展示了如何使用这个属性：

```
let iframe = document.getElementById("myIframe");
let iframeDoc = iframe.contentDocument;
```

contentDocument 属性是 Document 的实例，拥有所有文档属性和方法，因此可以像使用其他 HTML 文档一样使用它。还有一个属性 contentWindow，返回相应窗格的 window 对象，这个对象上有一个 document 属性。所有现代浏览器都支持 contentDocument 和 contentWindow 属性。

> **注意**　跨源访问子内嵌窗格的 document 对象会受到安全限制。如果内嵌窗格中加载了不同域名（或子域名）的页面，或者该页面使用了不同协议，则访问其 document 对象会抛出错误。

16.2 样式

HTML 中的样式有 3 种定义方式：外部样式表（通过<link>元素）、文档样式表（使用<style>元素）和元素特定样式（使用 style 属性）。DOM2 Style 为这 3 种应用样式的机制都提供了 API。

16.2.1 存取元素样式

任何支持 style 属性的 HTML 元素在 JavaScript 中都会有一个对应的 style 属性。这个 style 属性是 CSSStyleDeclaration 类型的实例，其中包含通过 HTML style 属性为元素设置的所有样式信息，但不包含通过层叠机制从文档样式和外部样式中继承来的样式。HTML style 属性中的 CSS 属性在 JavaScript style 对象中都有对应的属性。因为 CSS 属性名使用连字符表示法（用连字符分隔两个单词，如 background-image），所以在 JavaScript 中这些属性必须转换为驼峰大小写形式（如 backgroundImage）。下表给出了几个常用的 CSS 属性与 style 对象中等价属性的对比。

CSS 属性	JavaScript 属性
background-image	style.backgroundImage
color	style.color
display	style.display
font-family	style.fontFamily

大多数属性名会这样直接转换过来。但有一个 CSS 属性名不能直接转换，它就是 float。因为 float 是 JavaScript 的保留字，所以不能用作属性名。DOM2 Style 规定它在 style 对象中对应的属性应该是 cssFloat。

任何时候，只要获得了有效 DOM 元素的引用，就可以通过 JavaScript 来设置样式。来看下面的例子：

```
let myDiv = document.getElementById("myDiv");

// 设置背景颜色
myDiv.style.backgroundColor = "red";

// 修改大小
myDiv.style.width = "100px";
myDiv.style.height = "200px";

// 设置边框
myDiv.style.border = "1px solid black";
```

像这样修改样式时，元素的外观会自动更新。

> **注意** 在标准模式下，所有尺寸都必须包含单位。在混杂模式下，可以把 style.width 设置为"20"，相当于"20px"。如果是在标准模式下，把 style.width 设置为"20"会被忽略，因为没有单位。实践中，最好一直加上单位。

通过 style 属性设置的值也可以通过 style 对象获取。比如下面的 HTML：

```
<div id="myDiv" style="background-color: blue; width: 10px; height: 25px"></div>
```

这个元素 style 属性的值可以像这样通过代码获取：

```
console.log(myDiv.style.backgroundColor);   // "blue"
console.log(myDiv.style.width);             // "10px"
console.log(myDiv.style.height);            // "25px"
```

如果元素上没有 style 属性，则 style 对象包含所有可能的 CSS 属性的空值。

1. DOM 样式属性和方法

DOM2 Style 规范也在 style 对象上定义了一些属性和方法。这些属性和方法提供了元素 style 属性的信息并支持修改，列举如下。

- cssText，包含 style 属性中的 CSS 代码。
- length，应用给元素的 CSS 属性数量。
- parentRule，表示 CSS 信息的 CSSRule 对象（下一节会讨论 CSSRule 类型）。
- getPropertyCSSValue(*propertyName*)，返回包含 CSS 属性 *propertyName* 值的 CSSValue 对象（已废弃）。
- getPropertyPriority(*propertyName*)，如果 CSS 属性 *propertyName* 使用了!important 则返回"important"，否则返回空字符串。
- getPropertyValue(*propertyName*)，返回属性 *propertyName* 的字符串值。
- item(*index*)，返回索引为 *index* 的 CSS 属性名。
- removeProperty(*propertyName*)，从样式中删除 CSS 属性 *propertyName*。
- setProperty(*propertyName*, *value*, *priority*)，设置 CSS 属性 *propertyName* 的值为 *value*，*priority* 是"important"或空字符串。

通过 cssText 属性可以存取样式的 CSS 代码。在读模式下，cssText 返回 style 属性 CSS 代码在浏览器内部的表示。在写模式下，给 cssText 赋值会重写整个 style 属性的值，意味着之前通过 style 属性设置的属性都会丢失。比如，如果一个元素通过 style 属性设置了边框，而赋给 cssText 属性的值不包含边框，则元素的边框会消失。下面的例子演示了 cssText 的使用：

```
myDiv.style.cssText = "width: 25px; height: 100px; background-color: green";
console.log(myDiv.style.cssText);
```

设置 cssText 是一次性修改元素多个样式最快捷的方式，因为所有变化会同时生效。

length 属性是跟 item()方法一起配套迭代 CSS 属性用的。此时，style 对象实际上变成了一个集合，也可以用中括号代替 item()取得相应位置的 CSS 属性名，如下所示：

```
for (let i = 0, len = myDiv.style.length; i < len; i++) {
  console.log(myDiv.style[i]);   // 或者用 myDiv.style.item(i)
}
```

使用中括号或者 item()都可以取得相应位置的 CSS 属性名（"background-color"，不是"backgroundColor"）。这个属性名可以传给 getPropertyValue()以取得属性的值，如下面的例子所示：

```
let prop, value, i, len;
for (i = 0, len = myDiv.style.length; i < len; i++) {
  prop = myDiv.style[i];   // 或者用 myDiv.style.item(i)
  value = myDiv.style.getPropertyValue(prop);
  console.log(`prop: ${value}`);
}
```

getPropertyValue()方法返回 CSS 属性值的字符串表示。如果需要更多信息，则可以通过 getPropertyCSSValue()获取 CSSValue 对象。这个对象有两个属性：cssText 和 cssValueType。前者的值与 getPropertyValue()方法返回的值一样；后者是一个数值常量，表示当前值的类型（0 代表继承的值，1 代表原始值，2 代表列表，3 代表自定义值）。[①]下面的代码演示了如何输出 CSS 属性值和值类型：

```
let prop, value, i, len;
for (i = 0, len = myDiv.style.length; i < len; i++) {
  prop = myDiv.style[i];   // alternately, myDiv.style.item(i)
  value = myDiv.style.getPropertyCSSValue(prop);
  console.log(`prop: ${value.cssText} (${value.cssValueType})`);
}
```

removeProperty()方法用于从元素样式中删除指定的 CSS 属性。使用这个方法删除属性意味着会应用该属性的默认（从其他样式表层叠继承的）样式。例如，可以像下面这样删除 style 属性中设置的 border 样式：

```
myDiv.style.removeProperty("border");
```

在不确定给定 CSS 属性的默认值是什么的时候，可以使用这个方法。只要从 style 属性中删除，就可以使用默认值。

2. 计算样式

style 对象中包含支持 style 属性的元素为这个属性设置的样式信息，但不包含从其他样式表层叠继承的同样影响该元素的样式信息。DOM2 Style 在 document.defaultView 上增加了 getComputedStyle()方法。这个方法接收两个参数：要取得计算样式的元素和伪元素字符串（如":after"）。如果不需要查询伪元素，则第二个参数可以传 null。getComputedStyle()方法返回一个 CSSStyleDeclaration 对象（与 style 属性的类型一样），包含元素的计算样式。假设有如下 HTML 页面：

```
<!DOCTYPE html>
<html>
<head>
  <title>Computed Styles Example</title>
  <style type="text/css">
    #myDiv {
      background-color: blue;
      width: 100px;
      height: 200px;
    }
  </style>
</head>
<body>
  <div id="myDiv" style="background-color: red; border: 1px solid black"></div>
</body>
</html>
```

这里的 元素从文档样式表（<style>元素）和自己的 style 属性获取了样式。此时，这个元素的 style 对象中包含 backgroundColor 和 border 属性，但不包含（通过样式表规则应用的）width 和 height 属性。下面的代码从这个元素获取了计算样式：

---

① 不过，getPropertyCSSValue()方法已经被废弃，虽然可能有浏览器还支持，但随时有可能被删除。建议开发中使用 getPropertyValue()。——译者注

```
let myDiv = document.getElementById("myDiv");
let computedStyle = document.defaultView.getComputedStyle(myDiv, null);

console.log(computedStyle.backgroundColor);  // "red"
console.log(computedStyle.width);            // "100px"
console.log(computedStyle.height);           // "200px"
console.log(computedStyle.border);           // "1px solid black"（在某些浏览器中）
```

在取得这个元素的计算样式时，得到的背景颜色是"red"，宽度为"100px"，高度为"200px"。背景颜色不是"blue"，因为元素样式覆盖了它。border 属性不一定返回样式表中实际的 border 规则（某些浏览器会）。这种不一致性是因浏览器解释简写样式的方式造成的，比如 border 实际上会设置一组别的属性。在设置 border 时，实际上设置的是 4 条边的线条宽度、颜色和样式（border-left-width、border-top-color、border-bottom-style 等）。因此，即使 computedStyle.border 在所有浏览器中都不会返回值，computedStyle.borderLeftWidth 也一定会返回值。

> **注意**　浏览器虽然会返回样式值，但返回值的格式不一定相同。比如，Firefox 和 Safari 会把所有颜色值转换为 RGB 格式（如红色会变成 rgb(255,0,0)），而 Opera 把所有颜色转换为十六进制表示法（如红色会变成#ff0000）。因此在使用 getComputedStyle()时一定要多测试几个浏览器。

关于计算样式要记住一点，在所有浏览器中计算样式都是只读的，不能修改 getComputedStyle()方法返回的对象。而且，计算样式还包含浏览器内部样式表中的信息。因此有默认值的 CSS 属性会出现在计算样式里。例如，visibility 属性在所有浏览器中都有默认值，但这个值因实现而不同。有些浏览器会把 visibility 的默认值设置为"visible"，而另一些将其设置为"inherit"。不能假设 CSS 属性的默认值在所有浏览器中都一样。如果需要元素具有特定的默认值，那么一定要在样式表中手动指定。

16.2.2　操作样式表

CSSStyleSheet 类型表示 CSS 样式表，包括使用<link>元素和通过<style>元素定义的样式表。注意，这两个元素本身分别是 HTMLLinkElement 和 HTMLStyleElement。CSSStyleSheet 类型是一个通用样式表类型，可以表示以任何方式在 HTML 中定义的样式表。另外，元素特定的类型允许修改 HTML 属性，而 CSSStyleSheet 类型的实例则是一个只读对象（只有一个属性例外）。

CSSStyleSheet 类型继承 StyleSheet，后者可用作非 CSS 样式表的基类。以下是 CSSStyleSheet 从 StyleSheet 继承的属性。

- disabled，布尔值，表示样式表是否被禁用了（这个属性是可读写的，因此将它设置为 true 会禁用样式表）。
- href，如果是使用<link>包含的样式表，则返回样式表的 URL，否则返回 null。
- media，样式表支持的媒体类型集合，这个集合有一个 length 属性和一个 item()方法，跟所有 DOM 集合一样。同样跟所有 DOM 集合一样，也可以使用中括号访问集合中特定的项。如果样式表可用于所有媒体，则返回空列表。
- ownerNode，指向拥有当前样式表的节点，在 HTML 中要么是<link>元素要么是<style>元素（在 XML 中可以是处理指令）。如果当前样式表是通过@import 被包含在另一个样式表中，则这个属性值为 null。

- parentStyleSheet，如果当前样式表是通过@import 被包含在另一个样式表中，则这个属性指向导入它的样式表。
- title，ownerNode 的 title 属性。
- type，字符串，表示样式表的类型。对 CSS 样式表来说，就是"text/css"。

上述属性里除了 disabled，其他属性都是只读的。除了上面继承的属性，CSSStyleSheet 类型还支持以下属性和方法。

- cssRules，当前样式表包含的样式规则的集合。
- ownerRule，如果样式表是使用@import 导入的，则指向导入规则；否则为 null。
- deleteRule(*index*)，在指定位置删除 cssRules 中的规则。
- insertRule(*rule, index*)，在指定位置向 cssRules 中插入规则。

document.styleSheets 表示文档中可用的样式表集合。这个集合的 length 属性保存着文档中样式表的数量，而每个样式表都可以使用中括号或 item()方法获取。来看这个例子：

```
let sheet = null;
for (let i = 0, len = document.styleSheets.length; i < len; i++) {
  sheet = document.styleSheets[i];
  console.log(sheet.href);
}
```

以上代码输出了文档中每个样式表的 href 属性（<style>元素没有这个属性）。

document.styleSheets 返回的样式表可能会因浏览器而异。所有浏览器都会包含<style>元素和 rel 属性设置为"stylesheet"的<link>元素。IE、Opera、Chrome 也包含 rel 属性设置为"alternate stylesheet"的<link>元素。

通过<link>或<style>元素也可以直接获取 CSSStyleSheet 对象。DOM 在这两个元素上暴露了 sheet 属性，其中包含对应的 CSSStyleSheet 对象。

1. CSS 规则

CSSRule 类型表示样式表中的一条规则。这个类型也是一个通用基类，很多类型都继承它，但其中最常用的是表示样式信息的 CSSStyleRule（其他 CSS 规则还有@import、@font-face、@page 和@charset 等，不过这些规则很少需要使用脚本来操作）。以下是 CSSStyleRule 对象上可用的属性。

- cssText，返回整条规则的文本。这里的文本可能与样式表中实际的文本不一样，因为浏览器内部处理样式表的方式也不一样。Safari 始终会把所有字母都转换为小写。
- parentRule，如果这条规则被其他规则（如@media）包含，则指向包含规则，否则就是 null。
- parentStyleSheet，包含当前规则的样式表。
- selectorText，返回规则的选择符文本。这里的文本可能与样式表中实际的文本不一样，因为浏览器内部处理样式表的方式也不一样。这个属性在 Firefox、Safari、Chrome 和 IE 中是只读的，在 Opera 中是可以修改的。
- style，返回 CSSStyleDeclaration 对象，可以设置和获取当前规则中的样式。
- type，数值常量，表示规则类型。对于样式规则，它始终为 1。

在这些属性中，使用最多的是 cssText、selectorText 和 style。cssText 属性与 style.cssText 类似，不过并不完全一样。前者包含选择符文本和环绕样式声明的大括号，而后者则只包含样式声明（类似于元素上的 style.cssText）。此外，cssText 是只读的，而 style.cssText 可以被重写。

多数情况下，使用 style 属性就可以实现操作样式规则的任务了。这个对象可以像每个元素上的

style 对象一样，用来读取或修改规则的样式。比如下面这个 CSS 规则：

```
div.box {
  background-color: blue;
  width: 100px;
  height: 200px;
}
```

假设这条规则位于页面中的第一个样式表中，而且是该样式表中唯一一条 CSS 规则，则下列代码可以获取它的所有信息：

```
let sheet = document.styleSheets[0];
let rules = sheet.cssRules || sheet.rules;  // 取得规则集合
let rule = rules[0];                        // 取得第一条规则
console.log(rule.selectorText);             // "div.box"
console.log(rule.style.cssText);            // 完整的 CSS 代码
console.log(rule.style.backgroundColor);    // "blue"
console.log(rule.style.width);              // "100px"
console.log(rule.style.height);             // "200px"
```

使用这些接口，可以像确定元素 style 对象中包含的样式一样，确定一条样式规则的样式信息。与元素的场景一样，也可以修改规则中的样式，如下所示：

```
let sheet = document.styleSheets[0];
let rules = sheet.cssRules || sheet.rules;  // 取得规则集合
let rule = rules[0];                        // 取得第一条规则
rule.style.backgroundColor = "red"
```

注意，这样修改规则会影响到页面上所有应用了该规则的元素。如果页面上有两个 元素有"box"类，则这两个元素都会受到这个修改的影响。

2. 创建规则

DOM 规定，可以使用 insertRule()方法向样式表中添加新规则。这个方法接收两个参数：规则的文本和表示插入位置的索引值。下面是一个例子：

```
sheet.insertRule("body { background-color: silver }", 0);  // 使用 DOM 方法
```

这个例子插入了一条改变文档背景颜色的规则。这条规则是作为样式表的第一条规则（位置 0）插入的，顺序对规则层叠是很重要的。

虽然可以这样添加规则，但随着要维护的规则增多，很快就会变得非常麻烦。这时候，更好的方式是使用第 14 章介绍的动态样式加载技术。

3. 删除规则

支持从样式表中删除规则的 DOM 方法是 deleteRule()，它接收一个参数：要删除规则的索引。要删除样式表中的第一条规则，可以这样做：

```
sheet.deleteRule(0);  // 使用 DOM 方法
```

与添加规则一样，删除规则并不是 Web 开发中常见的做法。考虑到可能影响 CSS 层叠的效果，删除规则时要慎重。

16.2.3　元素尺寸

本节介绍的属性和方法并不是 DOM2 Style 规范中定义的，但与 HTML 元素的样式有关。DOM 一直缺乏页面中元素实际尺寸的规定。IE 率先增加了一些属性，向开发者暴露元素的尺寸信息。这些属性

现在已经得到所有主流浏览器支持。

1. 偏移尺寸

第一组属性涉及**偏移尺寸**（offset dimensions），包含元素在屏幕上占用的所有视觉空间。元素在页面上的视觉空间由其高度和宽度决定，包括所有内边距、滚动条和边框（但不包含外边距）。以下 4 个属性用于取得元素的偏移尺寸。

- `offsetHeight`，元素在垂直方向上占用的像素尺寸，包括它的高度、水平滚动条高度（如果可见）和上、下边框的高度。
- `offsetLeft`，元素左边框外侧距离包含元素左边框内侧的像素数。
- `offsetTop`，元素上边框外侧距离包含元素上边框内侧的像素数。
- `offsetWidth`，元素在水平方向上占用的像素尺寸，包括它的宽度、垂直滚动条宽度（如果可见）和左、右边框的宽度。

其中，`offsetLeft` 和 `offsetTop` 是相对于包含元素的，包含元素保存在 `offsetParent` 属性中。`offsetParent` 不一定是 `parentNode`。比如，`<td>`元素的 `offsetParent` 是作为其祖先的`<table>`元素，因为`<table>`是节点层级中第一个提供尺寸的元素。图 16-1 展示了这些属性代表的不同尺寸。

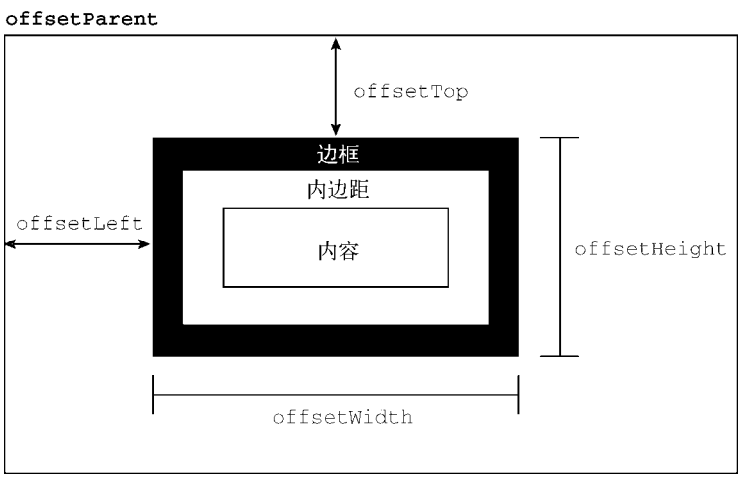

图 16-1

要确定一个元素在页面中的偏移量，可以把它的 `offsetLeft` 和 `offsetTop` 属性分别与 `offsetParent` 的相同属性相加，一直加到根元素。下面是一个例子：

```
function getElementLeft(element) {
  let actualLeft = element.offsetLeft;
  let current = element.offsetParent;

  while (current !== null) {
    actualLeft += current.offsetLeft;
    current = current.offsetParent;
  }

  return actualLeft;
}
```

```
function getElementTop(element) {
  let actualTop = element.offsetTop;
  let current = element.offsetParent;

  while (current !== null) {
    actualTop += current.offsetTop;
    current = current.offsetParent;
  }

  return actualTop;
}
```

这两个函数使用 offsetParent 在 DOM 树中逐级上溯，将每一级的偏移属性相加，最终得到元素的实际偏移量。对于使用 CSS 布局的简单页面，这两个函数是很精确的。而对于使用表格和内嵌窗格的页面布局，它们返回的值会因浏览器不同而有所差异，因为浏览器实现这些元素的方式不同。一般来说，包含在 元素中所有元素都以<body>为其 offsetParent，因此 getElementleft() 和 getElementTop()返回的值与 offsetLeft 和 offsetTop 返回的值相同。

> **注意**　所有这些偏移尺寸属性都是只读的，每次访问都会重新计算。因此，应该尽量减少查询它们的次数。比如把查询的值保存在局量中，就可以避免影响性能。

2. 客户端尺寸

元素的**客户端尺寸**（client dimensions）包含元素内容及其内边距所占用的空间。客户端尺寸只有两个相关属性：clientWidth 和 clientHeight。其中，clientWidth 是内容区宽度加左、右内边距宽度，clientHeight 是内容区高度加上、下内边距高度。图 16-2 形象地展示了这两个属性。

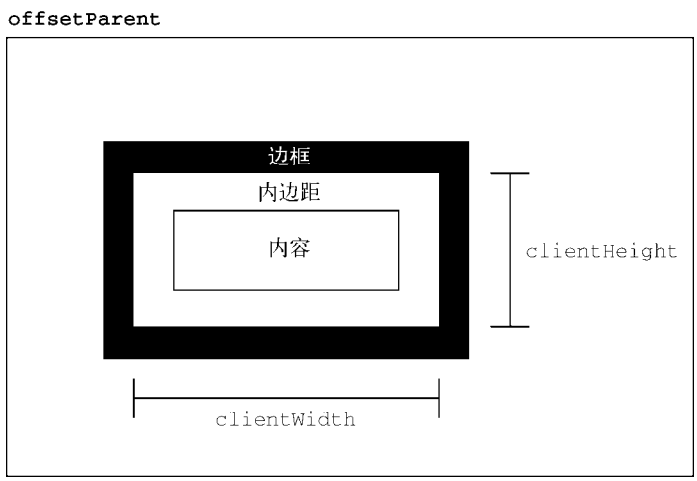

图　16-2

客户端尺寸实际上就是元素内部的空间，因此不包含滚动条占用的空间。这两个属性最常用于确定浏览器视口尺寸，即检测 document.documentElement 的 clientWidth 和 clientHeight。这两个属性表示视口（<html>或<body>元素）的尺寸。

> **注意** 与偏移尺寸一样，客户端尺寸也是只读的，而且每次访问都会重新计算。

3. 滚动尺寸

最后一组尺寸是**滚动尺寸**（scroll dimensions），提供了元素内容滚动距离的信息。有些元素，比如<html>无须任何代码就可以自动滚动，而其他元素则需要使用 CSS 的 overflow 属性令其滚动。滚动尺寸相关的属性有如下 4 个。

- scrollHeight，没有滚动条出现时，元素内容的总高度。
- scrollLeft，内容区左侧隐藏的像素数，设置这个属性可以改变元素的滚动位置。
- scrollTop，内容区顶部隐藏的像素数，设置这个属性可以改变元素的滚动位置。
- scrollWidth，没有滚动条出现时，元素内容的总宽度。

图 16-3 展示了这些属性的含义。

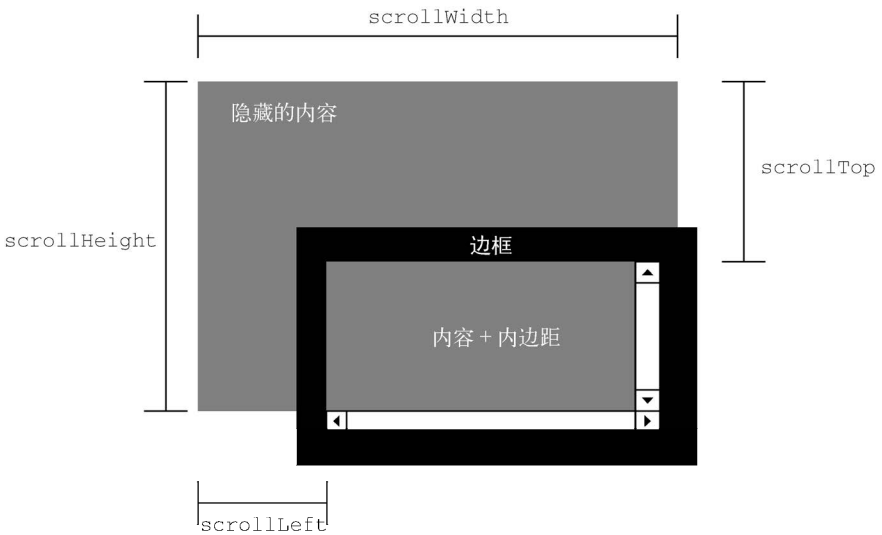

图 16-3

scrollWidth 和 scrollHeight 可以用来确定给定元素内容的实际尺寸。例如，<html>元素是浏览器中滚动视口的元素。因此，document.documentElement.scrollHeight 就是整个页面垂直方向的总高度。

scrollWidth 和 scrollHeight 与 clientWidth 和 clientHeight 之间的关系在不需要滚动的文档上是分不清的。如果文档尺寸超过视口尺寸，则在所有主流浏览器中这两对属性都不相等，scrollWidth 和 scollHeight 等于文档内容的宽度，而 clientWidth 和 clientHeight 等于视口的大小。

scrollLeft 和 scrollTop 属性可以用于确定当前元素滚动的位置，或者用于设置它们的滚动位置。元素在未滚动时，这两个属性都等于 0。如果元素在垂直方向上滚动，则 scrollTop 会大于 0，表示元素顶部不可见区域的高度。如果元素在水平方向上滚动，则 scrollLeft 会大于 0，表示元素左侧不可见区域的宽度。因为这两个属性也是可写的，所以把它们都设置为 0 就可以重置元素的滚动位置。