Text 节点和 DocumentFragment 节点

1. Text 节点的概念

文本节点（Text）代表元素节点（Element）和属性节点（Attribute）的文本内容。如果一个节点只包含一段文本，那么它就有一个文本子节点，代表该节点的文本内容。

通常我们使用父节点的 firstChild、nextSibling 等属性获取文本节点，或者使用 Document 节点的 createTextNode 方法创造一个文本节点。

```
// 获取文本节点
let textNode = document.querySelector('p').firstChild;
```

```
// 创造文本节点
let textNode = document.createTextNode('Hi');
document.querySelector('div').appendChild(textNode);
```

浏览器原生提供一个 Text 构造函数。它返回一个文本节点实例。它的参数就是该文本节点的文本内容。

```
// 空字符串
let text1 = new Text();
```

```
// 非空字符串
let text2 = new Text('This is a text node');
```

注意，由于空格也是一个字符，所以哪怕只有一个空格，也会形成文本节点。比如， 包含一个空格，它的子节点就是一个文本节点。

文本节点除了继承 Node 接口，还继承了 CharacterData 接口。Node 接口的属性和方法请参考《Node 接口》一章，这里不再重复介绍了，以下的属性和方法大部分来自 CharacterData 接口。

1.1. Text 节点的属性

1.1.1. data

data 属性等同于 nodeValue 属性，用来设置或读取文本节点的内容。

```
// 读取文本内容
document.querySelector('p').firstChild.data;
// 等同于
document.querySelector('p').firstChild.nodeValue;
```

```
// 设置文本内容
document.querySelector('p').firstChild.data = 'Hello World';
```

1.1.2. wholeText

`wholeText` 属性将当前文本节点与毗邻的文本节点，作为一个整体返回。大多数情况下，wholeText 属性的返回值，与 data 属性和 textContent 属性相同。但是，某些特殊情况会有差异。

举例来说，HTML 代码如下。

```
<p id="para">A <em>B</em> C</p>
```

这时，文本节点的 `wholeText` 属性和 `data` 属性，返回值相同。

```
let el = document.getElementById('para');
el.firstChild.wholeText; // "A "
el.firstChild.data; // "A "
```

但是，一旦移除 `<em>` 节点，`wholeText` 属性与 `data` 属性就会有差异，因为这时其实 `<p>` 节点下面包含了两个毗邻的文本节点。

```
el.removeChild(para.childNodes[1]);
el.firstChild.wholeText; // "A C"
el.firstChild.data; // "A "
```

1.1.3 length

`length` 属性返回当前文本节点的文本长度。

```
new Text('Hello').length; // 5
```

1.1.4. nextElementSibling 和 previousElementSibling

`nextElementSibling` 属性返回紧跟在当前文本节点后面的那个同级元素节点。如果取不到元素节点，则返回 `null`。

```
// HTML 为
// <div>Hello <em>World</em></div>
```

```
let tn = document.querySelector('div').firstChild;
tn.nextElementSibling;
// <em>World</em>
```

`previousElementSibling` 属性返回当前文本节点前面最近的同级元素节点。如果取不到元素节点，则返回 `null`。

1.2. Text 节点的方法

以下 5 个方法都是编辑 Text 节点文本内容的方法。

- `appendData()`：在 Text 节点尾部追加字符串。
- `deleteData()`：删除 Text 节点内部的子字符串，第一个参数为子字符串开始位置，第二个参数为子字符串长度。
- `insertData()`：在 Text 节点插入字符串，第一个参数为插入位置，第二个参数为插入的子字符串。
- `replaceData()`：用于替换文本，第一个参数为替换开始位置，第二个参数为需要被替换掉的长度，第三个参数为新加入的字符串。
- `subStringData()`：用于获取子字符串，第一个参数为子字符串在 Text 节点中的开始位置，第二个参数为子字符串长度。

```
// HTML 代码为
// <p>Hello World</p>
let pElementText = document.querySelector('p').firstChild;

pElementText.appendData('!');
// 页面显示 Hello World!
pElementText.deleteData(7, 5);
// 页面显示 Hello W
pElementText.insertData(7, 'Hello ');
// 页面显示 Hello WHello
pElementText.replaceData(7, 5, 'World');
// 页面显示 Hello WWorld
pElementText.substringData(7, 10);
// 页面显示不变，返回"World "
```

1.2.1 remove()

`remove()` 方法用于移除当前 `Text` 节点。

```
// HTML 代码为
// <p>Hello World</p>
document.querySelector('p').firstChild.remove();
// 现在 HTML 代码为
// <p></p>
```

1.2.2. splitText()

`splitText` 方法将 `Text` 节点一分为二，变成两个毗邻的 `Text` 节点。它的参数就是分割位置（从零开始），分割到该位置的字符前结束。如果分割位置不存在，将报错。

分割后，该方法返回分割位置后方的字符串，而原 Text 节点变成只包含分割位置前方的字符串。

```
// html 代码为 <p id="p">foobar</p>
let p = document.getElementById('p');
let textnode = p.firstChild;

let newText = textnode.splitText(3);
newText; // "bar"
textnode; // "foo"
```

父元素节点的 `normalize` 方法可以将毗邻的两个 `Text` 节点合并。

接上面的例子，文本节点的 `splitText` 方法将一个 `Text` 节点分割成两个，父元素的 `normalize` 方法可以实现逆操作，将它们合并。

```
p.childNodes.length; // 2

// 将毗邻的两个 Text 节点合并
p.normalize();
p.childNodes.length; // 1
```

2. DocumentFragment 节点

`DocumentFragment` 节点代表一个文档的片段，本身就是一个完整的 DOM 树形结构。它没有父节点，`parentNode` 返回 `null`，但是可以插入任意数量的子节点。它不属于当前文档，操作 `DocumentFragment` 节点，要比直接操作 `DOM` 树快得多。

它一般用于构建一个 `DOM` 结构，然后插入当前文档。`document.createDocumentFragment()` 方法，以及浏览器原生的 `DocumentFragment` 构造函数，可以创建一个空的 `DocumentFragment` 节点。然后再使用其他 `DOM` 方法，向其添加子节点。

```
let docFrag = document.createDocumentFragment();
// 等同于
let docFrag = new DocumentFragment();

let li = document.createElement('li');
li.textContent = 'Hello World';
docFrag.appendChild(li);

document.querySelector('ul').appendChild(docFrag);
```

上面代码创建了一个 `DocumentFragment` 节点，然后将一个 `li` 节点添加在它里面，最后将 `DocumentFragment` 节点移动到原文档。

注意，DocumentFragment 节点本身不能被插入当前文档。当它作为 appendChild()、insertBefore()、replaceChild() 等方法的参数时，是它的所有子节点插入当前文档，而不是它自身。一旦 DocumentFragment 节点被添加进当前文档，它自身就变成了空节点（textContent 属性为空字符串），可以被再次使用。如果想要保存 DocumentFragment 节点的内容，可以使用 cloneNode 方法。

```
document.querySelector('ul').appendChild(docFrag.cloneNode(true));
```

上面这样添加 DocumentFragment 节点进入当前文档，不会清空 DocumentFragment 节点。

下面是一个例子，使用 DocumentFragment 反转一个指定节点的所有子节点的顺序。

```
function reverse(n) {
  let f = document.createDocumentFragment();
  while (n.lastChild) f.appendChild(n.lastChild);
  n.appendChild(f);
}
```

DocumentFragment 节点对象没有自己的属性和方法，全部继承自 Node 节点和 ParentNode 接口。也就是说，DocumentFragment 节点比 Node 节点多出以下四个属性。

- children：返回一个动态的 HTMLCollection 集合对象，包括当前 DocumentFragment 对象的所有子元素节点。
- firstElementChild：返回当前 DocumentFragment 对象的第一个子元素节点，如果没有则返回 null。
- lastElementChild：返回当前 DocumentFragment 对象的最后一个子元素节点，如果没有则返回 null。
- childElementCount：返回当前 DocumentFragment 对象的所有子元素数量。