Node 节点集合

节点都是单个对象，有时需要一种数据结构，能够容纳多个节点。DOM 提供两种节点集合，用于容纳多个节点：`NodeList` 和 `HTMLCollection` 。

这两种集合都属于接口规范。许多 DOM 属性和方法，返回的结果是 `NodeList` 实例或 `HTMLCollection` 实例。区别是 `NodeList` 可以包含多种类型的节点，而 `HTMLCollection` 只能包含 HTML 元素。

1. NodeList 接口

`NodeList` 实例是一个类似数组的对象，它的成员是节点对象。通过以下方法可以得到 `NodeList` 实例。

- Node.childNodes
- document.querySelectorAll()

```
document.body.childNodes  // NodeList(15) [...]
document.querySelectorAll("div")  // NodeList(18) [...]
```

`NodeList` 实例很像数组，可以使用 `length` 属性和 `forEach` 方法。但是，它不是数组，不能使用 `pop` 或 `push` 之类数组特有的方法。

```
document.body.childNodes.length  // 15
document.querySelectorAll("div") instanceof NodeList // true
document.querySelectorAll("div").forEach(() => console.log(1))  // 18 次 1
```

如果 `NodeList` 实例要使用数组方法，可以将其转为真正的数组。

```
let children = document.body.childNodes;

Array.isArray(children);  // false

Array.isArray(Array.prototype.slice.call(children));  // true
```

除了使用 `forEach` 方法遍历 `NodeList` 实例，还可以使用 `for` 循环。

```
let children = document.body.childNodes;

for (let i = 0, len = children.length; i < len; i++) {
  let item = children[i];
}
```

`NodeList` **实例可能是动态集合，也可能是静态集合。** 所谓动态集合就是一个活的集合，`DOM` 删除或新增一个相关节点，都会立刻反映在 `NodeList` 实例。目前，只有 `Node.childNodes` 返回的是一个动态集合，其他的 `NodeList` 都是静态集合。

```
let childNode = document.body.childNodes;
childNode.length // 15

document.body.appendChild(document.createElement("p"))
childNode.length // 16

childNode[childNode.length - 1]  // <p></p>
typeof childNode[childNode.length - 1]  // "object"
```

1.1. NodeList.prototype.keys(), NodeList.prototype.value(), NodeList.prototype.entries()

这三个方法都返回一个 ES6 的遍历器对象，可以通过 `for...of` 循环遍历获取每一个成员的信息。区别在于，`keys()` 返回键名的遍历器，`values()` 返回键值的遍历器，`entries()` 返回的遍历器同时包含键名和键值的信息。

```
let children = document.body.childNodes;

for (let key of children.keys()) {
  console.log(key);
}
// 0
// 1
// 2
// ...

for (let value of children.values()) {
  console.log(value);
}
// #text
// <script>
// ...

for (let entry of children.entries()) {
  console.log(entry);
}
// Array [ 0, #text ]
// ...
// Array [ 15, p ]
// ...
```

2. HTMLCollection 接口

`HTMLCollection` 是一个节点对象的集合，只能包含元素节点（`element`），不能包含其他类型的节点。它的返回值是一个类似数组的对象，但是与 `NodeList` 接口不同，`HTMLCollection` 没有`forEach` 方法，只能使用

for 循环遍历。

HTMLCollection 实例都是动态集合，节点的变化会实时反映在集合中。

HTMLCollection 实例可以通过下面方法获得。

- document.getElementsByTagName("div")
- document.getElementsByClassName("className")
- document.links
- document.forms
- document.images
- document.scripts
- document.styleSheets

```
document.links instanceof HTMLCollection  // true
```

如果元素节点有 id 或 name 属性，那么 HTMLCollection 实例上面，可以使用 id 属性或 name属性引用该节点元素。如果没有对应的节点，则返回 null。

获取图片中的 id

```
// <img src="http://ip:port/logo.png" alt="logo" id="logo" />
console.log(document.getElementById("logo") === document.images.logo);  // true
console.log(document.getElementById("logo") instanceof HTMLImageElement);  //true
```

获取链接中的 name

```
// <a href="https://www.baidu.com" title="baidu" name="baidu">百度</a>
console.log(document.links.baidu === document.getElementsByName("baidu")[0]);  //
true
console.log(document.getElementsByName("baidu")[0] instanceof HTML);  //true
console.log(document.links.baidu instanceof HTMLLinkElement);  // false
console.log(document.links.baidu instanceof HTMLAnchorElement);  // true
```

HTMLLinkElement 接口表示外部资源的参考信息以及这些资源与文档的关系，反之亦然。这个对象继承了 HTMLElement 接口的所有属性和方法。

HTMLAnchorElement 接口表示超链接元素，并提供一些特别的属性和方法（除了那些继承自普通 HTMLElement 对象接口的之外）以用于操作这些元素的布局和显示。

2.1. HTMLCollection.prototype.namedItem()

namedItem 方法的参数是一个字符串，表示 id 属性或 name 属性的值，返回当前集合中对应的元素节点。如果没有对应的节点，则返回null。

```
// <img id="pic" src="http://example.com/foo.jpg">

var pic = document.getElementById('pic');
document.images.namedItem('pic') === pic // true
```

`Collection.namedItem('value')` 等同于 `Collection['value']` 。