



第16课: DOM对象进阶

■ 主讲老师: 万章



目录

DOM的核心概念

DOM的NodeList

DOM的HTMLCollection

DOM 的操作技术



DOM的核心概念

DOM的核心概念

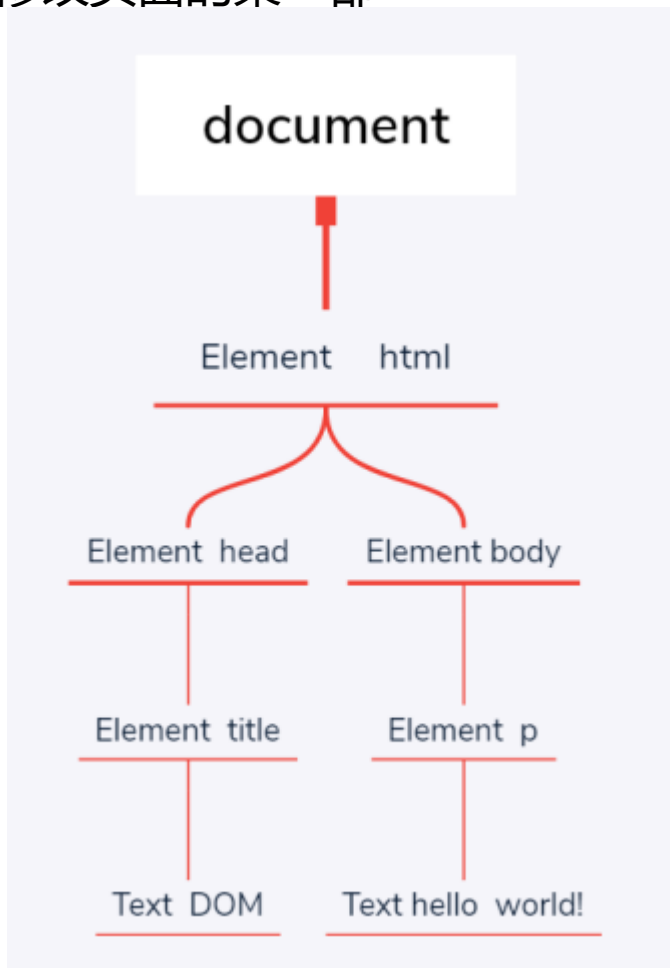
DOM（文档对象模型）是针对HTML 和XML 文档的一个API（应用程序编程接口）。DOM描绘了一个层次化的**节点树**，允许开发人员添加、移除和修改页面的某一部分

节点分为**几种不同的类型**，每种类型分别表示文档中不同的信息及（或）标记。每个节点都拥有各自的特点、数据和方法，另外也与其他节点存在某种关系。节点之间的关系构成了层次，而所有页面标记则表现为一个以特定节点为根节点的树形结构

文档节点只有一个子节点，即<html>元素，我们称之为文档元素。文档元素是文档的最外层元素，文档中的其他所有元素都包含在文档元素中。

每个文档只能有一个文档元素。在HTML 页面中，文档元素始终都是<html>元素

```
<html>
<head>
  <title>DOM</title>
</head>
<body>
  <p>Hello World!</p>
</body>
</html>
```



文档(document)节点是每个文档的根节点



DOM的NodeList

DOM元素的获取方式

```
> ele1
< ▼ NodeList(5) [li, li, li, li, li] ⓘ
  ▶ 0: li
  ▶ 1: li
  ▶ 2: li
  ▶ 3: li
  ▶ 4: li
  length: 5
  ▼ __proto__: NodeList
    ▶ entries: f entries()
    ▶ forEach: f forEach()
    ▶ item: f item()
    ▶ keys: f keys()
    length: (...)
    ▶ values: f values()
    ▶ constructor: f NodeList()
    ▶ Symbol(Symbol.iterator): f values()
    Symbol(Symbol.toStringTag): "NodeList"
    ▶ get length: f length()
    ▶ __proto__: Object

> ele2
< ▼ HTMLCollection(5) [li, li, li, li, li] ⓘ
  ▶ 0: li
  ▶ 1: li
  ▶ 2: li
  ▶ 3: li
  ▶ 4: li
  length: 5
  ▼ __proto__: HTMLCollection
    ▶ item: f item()
    length: (...)
    ▶ namedItem: f namedItem()
    ▶ constructor: f HTMLCollection()
    ▶ Symbol(Symbol.iterator): f values()
    Symbol(Symbol.toStringTag): "HTMLCollection"
    ▶ get length: f length()
    ▶ __proto__: Object
```

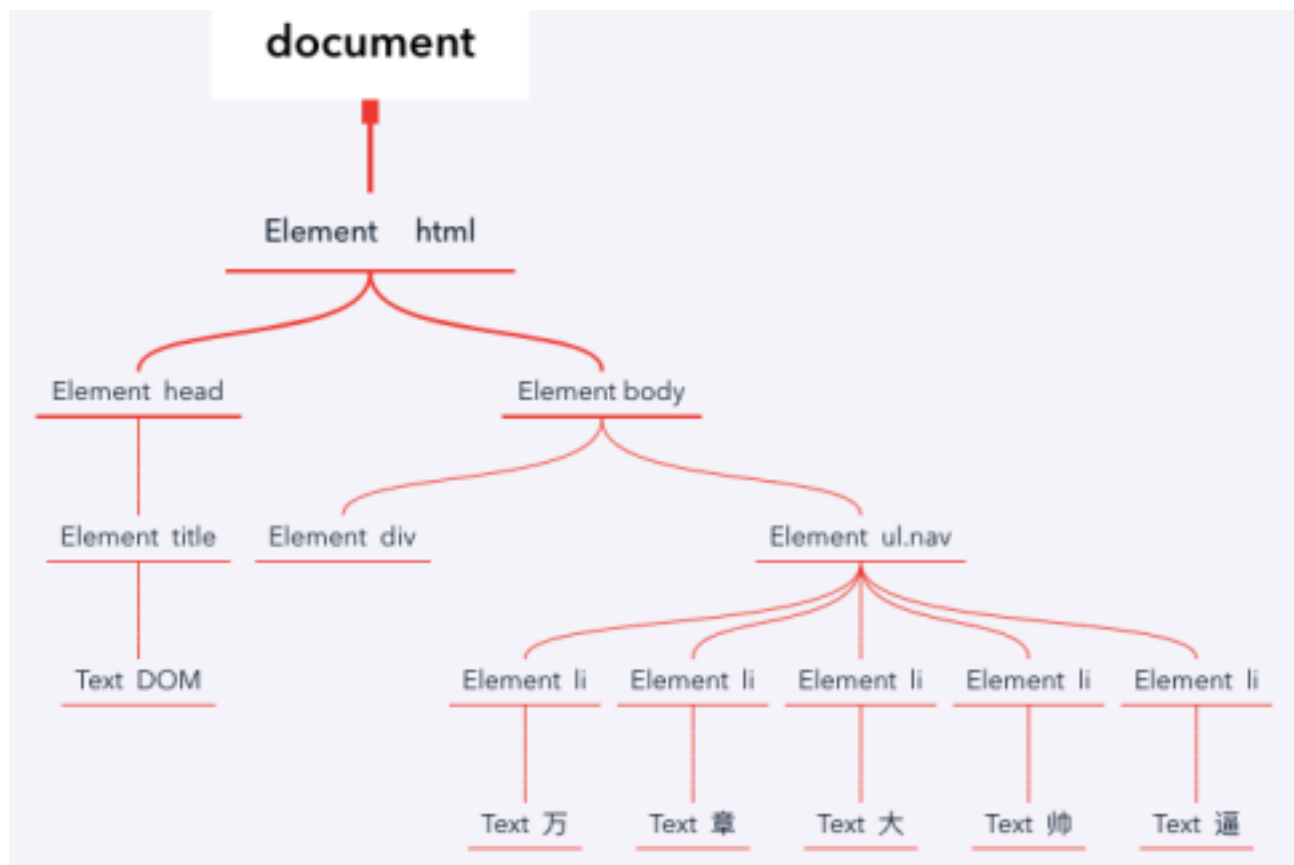
```
<div class="nav"></div>
<ul class="msg">
  <li>万</li>
  <li>章</li>
  <li>大</li>
  <li>帅</li>
  <li>逼</li>
</ul>

<script>
  let ele1=document.querySelectorAll("li");
  let ele2=document.getElementsByTagName("li");
</script>
```

在JavaScript的世界，获取元素的方式很多，但是获取的结果却分为两个明显的类别NodeList和HTMLCollection

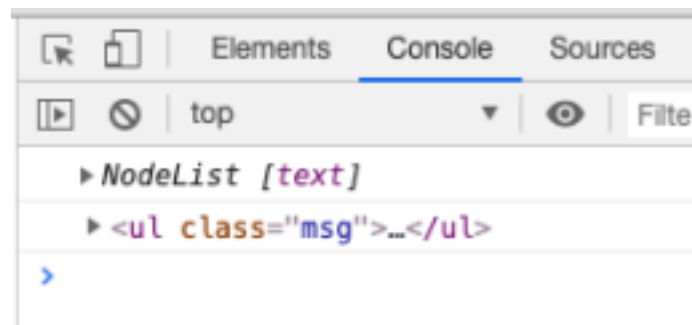
这两类都属于对象数据类型的一个分支，但是继承的默认方法有所不同

DOM节点层级之父子层级



```
<div class="nav"></div>
<ul class="msg">
  <li>万</li>
  <li>章</li>
  <li>大</li>
  <li>帅</li>
  <li>逼</li>
</ul>

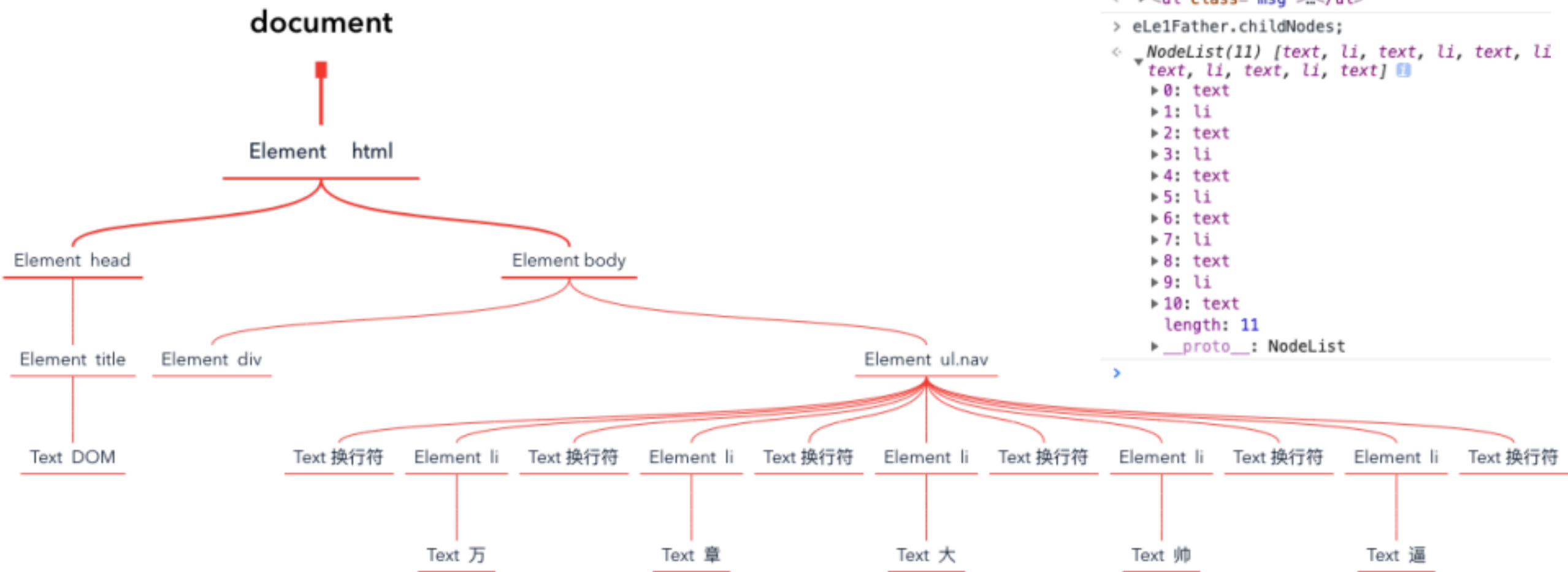
<script>
  let ele1=document.querySelectorAll("li");
  let ele1Son=ele1[0].childNodes;
  let ele1Father=ele1[0].parentNode;
  console.log(ele1Son);
  console.log(ele1Father);
</script>
```



childNodes:获取子节点（返回的是一个全新的NodeList，包含文本节点）

parentNode:获取父节点（返回的是父元素本身）

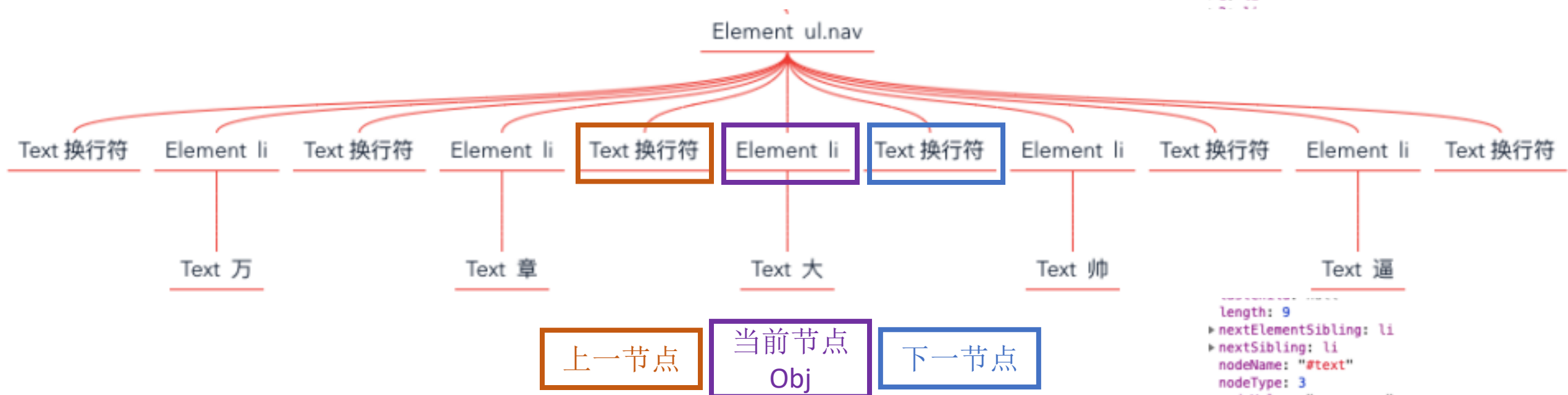
DOM节点层级之父子层级



```
Elements Console Sources >> | ⋮
top in the page to | Filter Default
> eLe1Father
< ▶ <ul class="msg">...</ul>
> eLe1Father.childNodes;
< ▼ NodeList(11) [text, li, text, li, text, li, text, li, text, li, text]
  ▶ 0: text
  ▶ 1: li
  ▶ 2: text
  ▶ 3: li
  ▶ 4: text
  ▶ 5: li
  ▶ 6: text
  ▶ 7: li
  ▶ 8: text
  ▶ 9: li
  ▶ 10: text
      length: 11
  ▶ __proto__: NodeList
>
```

牢记，子节点中文字和符号都会成为其中的一个节点！！！！

DOM节点层级之兄弟层级



```
Elements Console Sources >> >
top Filter Defau >

> ele1
< > ▼ NodeList(5) [li, li, li, li, li]
  > 0: li
  > 1: li
  > 2: li
  > 3: li
  > 4: li

length: 9
nextElementSibling: li
nextSibling: li
nodeName: "#text"
nodeType: 3
nodeValue: "
"
ownerDocument: document
parentElement: ul.msg
parentNode: ul.msg
previousElementSibling: li
previousSibling: li
textContent: "
"
wholeText: "
"
__proto__: Text

> ele1[2].previousSibling
< > #text

> ele1Father.childNodes[0];
< > #text

> ele1Father.childNodes[0].previousSibling;
< > null

> ele1Father.childNodes[10].nextSibling;
< > null

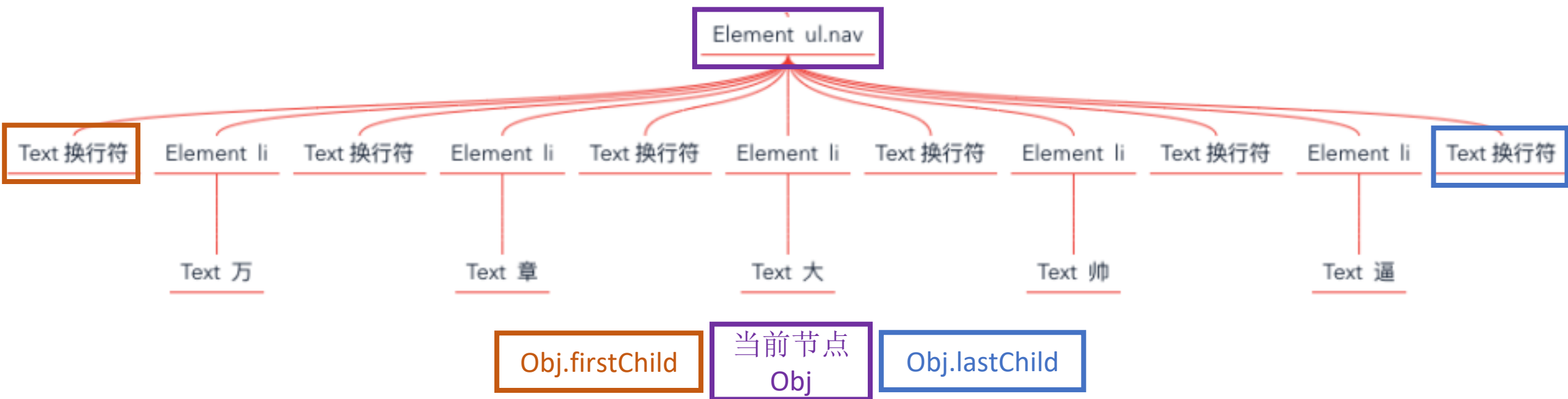
>
```

`Obj.nextSibling`:当前节点的同一层级的下一个兄弟节点, 如果当前节点已经是最后一个, 那么下一个节点则返回`null`

`Obj.previousSibling`:当前节点的同一层级的上一个兄弟节点, 如果当前节点已经是第一个, 那么上一个节点则返回`null`

同意层级只有一个节点, 那么该节点的 `nextSibling` 和 `previousSibling` 都为 `null`

DOM节点层级之父子层级



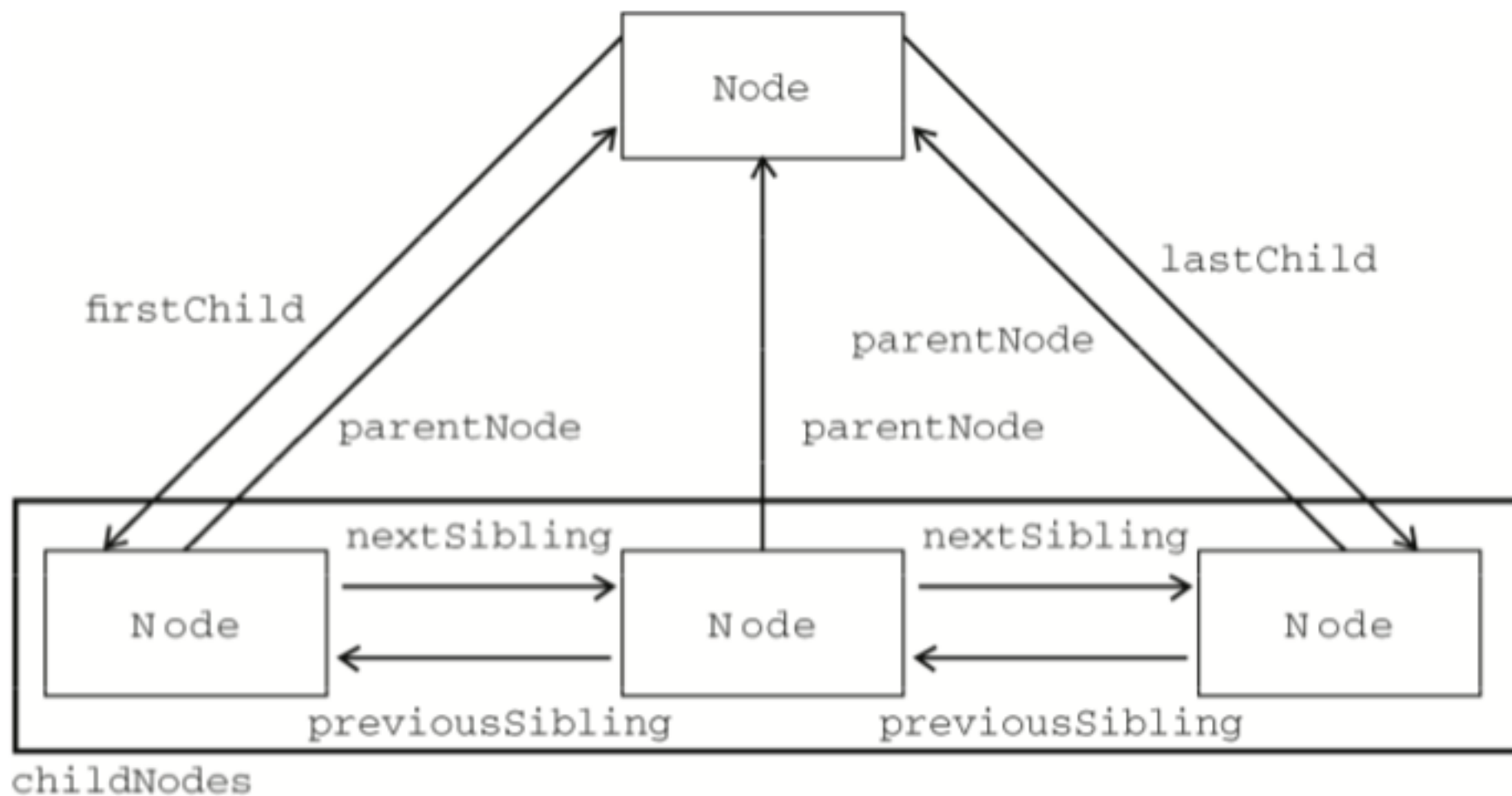
Obj.firstChild: 当前节点的子层级中的第一个节点

Obj.lastChild: 当前节点的子层级中的最后一个节点

在只有一个子节点的情况下，firstChild 和lastChild 指向同一个节点。

如果没有子节点，那么 firstChild 和 lastChild 的值均为 null

DOM的层级关系图

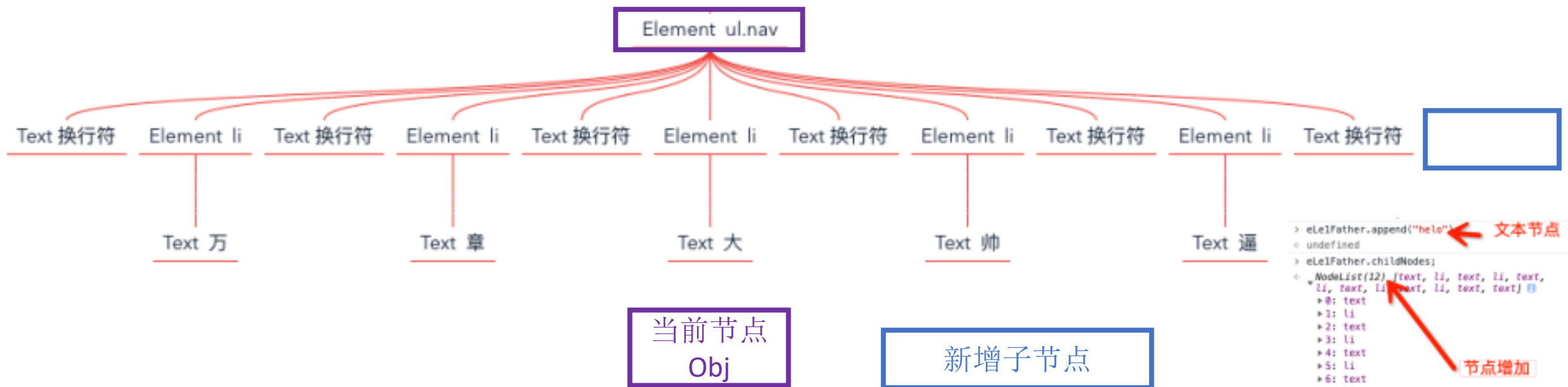


DOM的子节点判断 .hasChildNodes

{父} • hasChildNodes()

如果父节点内拥有子节点，返回true；如果没有子节点则返回false
比如文本节点, 属性节点就没有子节点

DOM的元素子节点插入appendChild ()

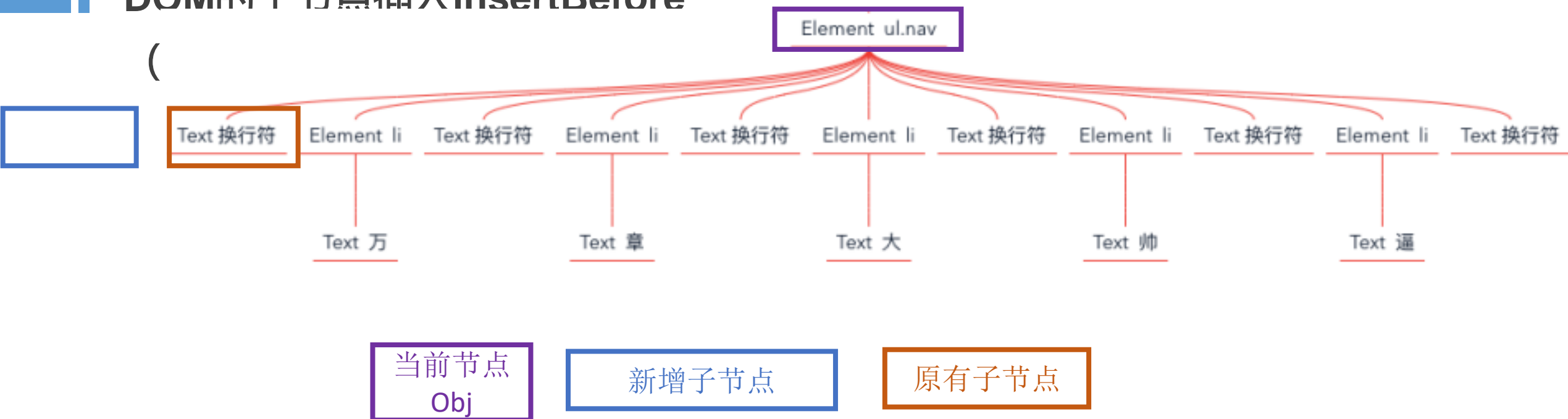


```
> e1Father.append("helo") ← 文本节点
< undefined
> e1Father.childNodes;
< NodeList(12) (text, li, text, li, text,
  * li, text, li, text, li, text, text)
  * 0: text
  * 1: li
  * 2: text
  * 3: li
  * 4: text
  * 5: li
  * 6: text
  * 7: li
  * 8: text
  * 9: li
  * 10: text
  * 11: text
    assignedSlot: null
    baseURI: "file:///Volumes/Data/4E54B7...
    * childNodes: NodeList []
    data: "helo"
    firstChild: null
    isConnected: true
    lastChild: null
    length: 4
    nextElementSibling: null
    nextSibling: null
    nodeName: "#text"
    * nodeType: 3
    * nodeValue: "helo" ← 节点增加
    * ownerDocument: document
    * parentElement: ul.msg
    * parentNode: ul.msg
    * previousElementSibling: li
    * previousSibling: text
    * textContent: "helo"
    * wholeText: " helo"
    * __proto__: Text
    length: 12
    * __proto__: NodeList
```

{父} • appendChild(子节点)

给某父节点内添加一个新的节点，添加的节点位于所有的子节点的最末
此处的子节点可以直接是一个字符串！！！！

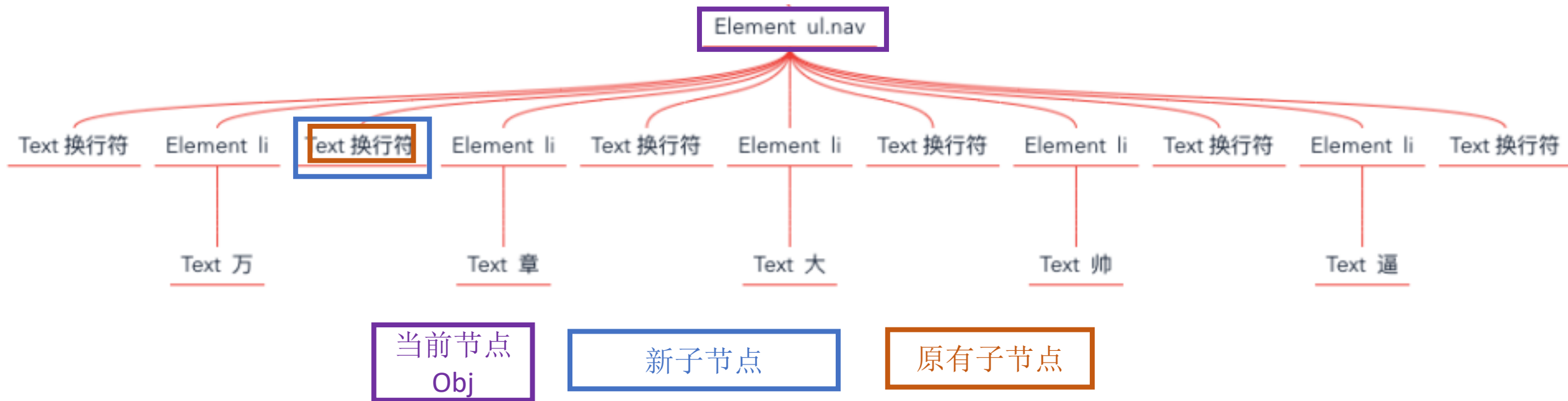
DOM的子节点插入insertBefore



{父} • insertBefore(新增子节点, 原有子节点)

给某父节点内的某一个子节点前添加一个新的子节点，添加的节点位于原有的子节点之前
此处的子节点不可以直接是一个字符串！！！！

DOM的子节点替换replaceChild ()



{父} • replaceChild(新子节点, 原有子节点)

把某父节点内的某一个子节点**替换成**一个新的子节点，添加的节点位于原有的子节点之前
此处的新子节点不可以直接是一个字符串！！！！



DOM的HTMLCollection

DOM的HTMLCollection概念

```
Elements Console >>
top Filter Defau
> ele
< HTMLCollection [ul.msg]
  ▶ 0: ul.msg
    length: 1
    __proto__: HTMLCollection
> eleSon
< NodeList(11) [text, li, text, li, text, li, text, li, text, li, text]
  ▶ 0: text
  ▶ 1: li
  ▶ 2: text
  ▶ 3: li
  ▶ 4: text
  ▶ 5: li
  ▶ 6: text
  ▶ 7: li
  ▶ 8: text
  ▶ 9: li
  ▶ 10: text
    length: 11
    __proto__: NodeList
> eLeFather
< <body>...</body>
```

```
<div class="nav"></div>
<ul class="msg">
  <li>万</li>
  <li>章</li>
  <li>大</li>
  <li>帅</li>
  <li>逼</li>
</ul>
<script>
  let ele=document.getElementsByTagName("ul");
  let eleSon=ele[0].childNodes;
  let eLeFather=ele[0].parentNode;
</script>
```

HTMLCollection内没有文本节点， 可以简单的理解为HTMLCollection就是木有文本节点的`nodeList`（当然有些方法不太一样），所以，HTMLCollection里面的元素也是可以使用`nodeList`的方法的



DOM的动态集合概念

1.NodeList

2.HTMLCollection

这两个集合是“动态的”，因此只要有新<div>元素被添加到页面中，这个元素也会被添加到该集合中。

浏览器在下一次访问集合时再更新集合



DOM 的操作技术

DOM的元素遍历



```
> ele[0].childElementCount
< 5
> ele[0].firstElementChild
< <li>万</li>
> ele[0].lastElementChild
< <li>逼</li>
> ele[0].firstElementChild.nextElementSibling
< <li>章</li>
> ele[0].lastElementChild.previousElementSibling
< <li>帅</li>
> |
```

```
<div class="nav"></div>
<ul class="msg">
  <li>万</li>
  <li>章</li>
  <li>大</li>
  <li>帅</li>
  <li>逼</li>
</ul>
<script>
  let ele=document.getElementsByTagName("ul");
  let eleSon=ele[0].childNodes;
  let eLeFather=ele[0].parentNode;
</script>
```

childElementCount: 返回子元素（不包括文本节点和注释）的个数。

firstElementChild: 指向第一个子元素；firstChild 的元素版。

lastElementChild: 指向最后一个子元素；lastChild 的元素版。

previousElementSibling: 指向前一个同辈元素；previousSibling 的元素版。

nextElementSibling: 指向后一个同辈元素；nextSibling 的元素版。

DOM的元素删除

{父} • removeChild(子节点)

删除父元素里面的某个子节点，此处的子节点可以是文本节点

注意：此处的子节点一定得是父元素的子节点，如果不是或是仅仅数据相同而地址不同的话，则会出错

{子节点} • remove ()

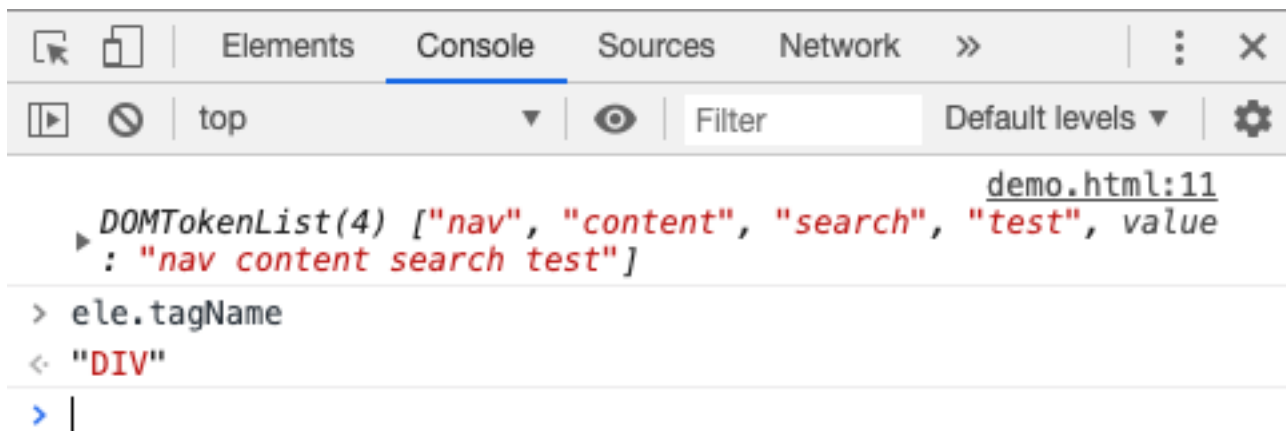
删除父元素里面的某个子节点，此处的子节点可以是文本节点

DOM的类名和标签名的获取

`obj.tagName;`

获取元素的标签名称， 注意:该方法获得的标签名称都是大写的

```
<div class="nav content search test"></div>
<script>
  let ele=document.querySelector(".nav");
  console.log(ele.classList);
</script>
```



`obj.classList;`

获取元素的类名数组， 注意:该方法获得的是元素的所有名称， 每一个名字都是数组里面的一个数组项目

`add(value)`: 将给定的字符串值添加到列表中。如果值已经存在，就不添加了。

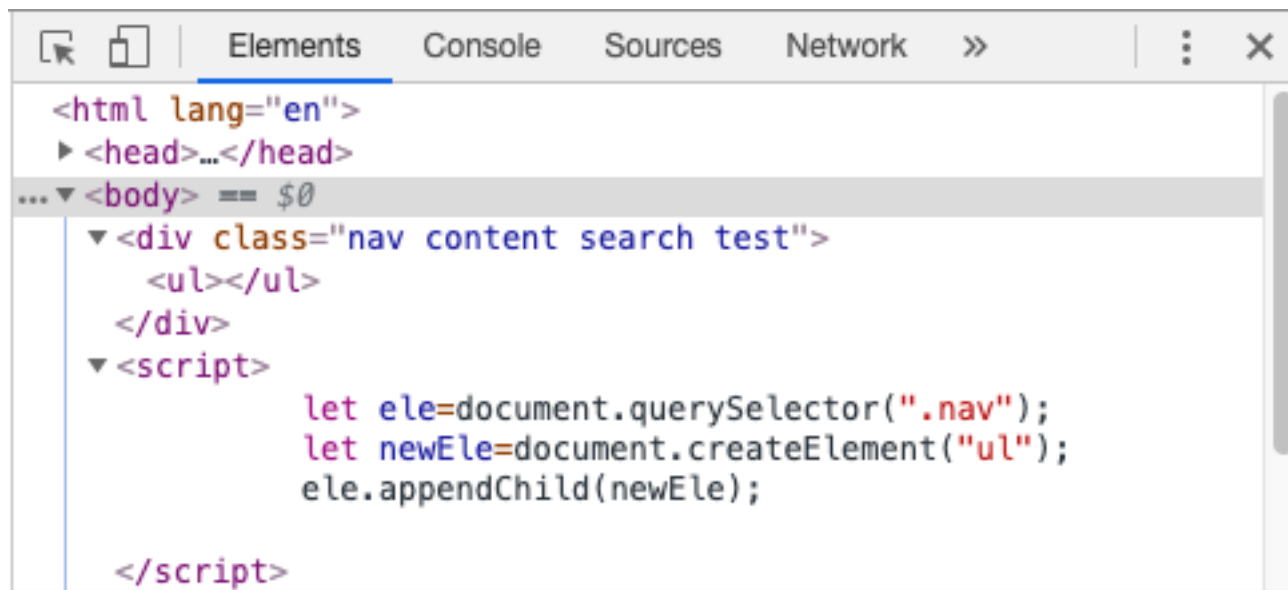
`contains(value)`: 表示列表中是否存在给定的值，如果存在则返回 `true`，否则返回 `false`。

`remove(value)`: 从列表中删除给定的字符串。

`toggle(value)`: 如果列表中已经存在给定的值，删除它；如果列表中没有给定的值，添加

DOM的单元元素创造

```
<div class="nav content search test"></div>
<script>
  let ele=document.querySelector(".nav");
  let newEle=document.createElement("ul");
  ele.appendChild(newEle);
</script>
```



document.createElement("元素的标签名称");
创造的元素默认是放在内存空间，不会在网页上显示，如果想让它在网页上显示，那么就要通过DOM方法，把元素加入到网页上来。
比如appendChild或是insertBefore等

DOM的元素片段创造

```
<html lang="en">
  <head>...</head>
  <body> == $0
    <div class="nav content search test">
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
    </div>
    <script>
      let ele=document.querySelector(".nav");
      for(let i=0;i<10;i++){
        let newEle=document.createElement("ul");
        ele.appendChild(newEle);//每次循环, 创造1个元素并添
        加到网页上
      }
    </script>
```

```
Elements Console Sources Network »
<html lang="en">
  <head>...</head>
  <body> == $0
    <div class="nav content search test">
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
      <ul></ul>
    </div>
    <script>
      let ele=document.querySelector(".nav");
      let eleArea=document.createDocumentFragment();//创造
      元素片段区域
      for(let i=0;i<10;i++){
        let newEle=document.createElement("ul");
        eleArea.appendChild(newEle);//每次循环, 创造1个元素
        并添加到网页上
      }
      ele.appendChild(eleArea);
    </script>
```

如果每次创造一个元素就把该元素添加到网页上, 那么会造成非常大的网页性能损耗。

所以我们可以先在内存中建立一个元素片段区域, 先把创造的元素添加到这个区域内, 再一次添加进网页中这样的性能会好的多

DOM的元素片段创造

内存数据区

Js生成的元素

Js生成的元素

...

Js生成的元素

生成一个
添加一个

内存数据区

网页元素

内存数据区

Js生成的元素

Js生成的元素

...

Js生成的元素

生成的元素
先预存起来

Js元素片段
Document
Fragment

一口气添加

内存数据区

网页元素

