Раздел
Документ

Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого

Задачи (4)
Комментарии

Свойства узлов: тип, тег и содержимое

Теперь давайте более внимательно взглянем на DOM-узлы.

В этой главе мы подробнее разберём, что они собой представляют и изучим их основные свойства.

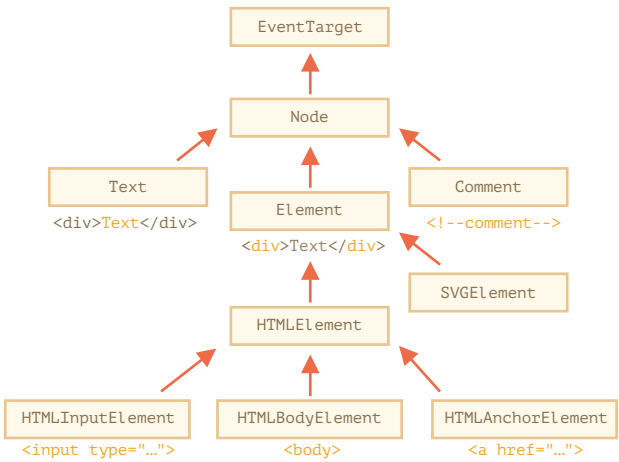
Классы DOM-узлов

У разных DOM-узлов могут быть разные свойства. Например, у узла, соответствующего тегу `<a>`, есть свойства, связанные со ссылками, а у соответствующего тегу `<input>` – свойства, связанные с полем ввода и т.д. Текстовые узлы отличаются от узлов-элементов. Но у них есть общие свойства и методы, потому что все классы DOM-узлов образуют единую иерархию.

Каждый DOM-узел принадлежит соответствующему встроенному классу.

Корнем иерархии является EventTarget, от него наследует Node и остальные DOM-узлы.

На рисунке ниже изображены основные классы:



Существуют следующие классы:

- EventTarget – это корневой «абстрактный» класс. Объекты этого класса никогда не создаются. Он служит основой, благодаря которой все DOM-узлы поддерживают так называемые «события», о которых мы поговорим позже.
- Node – также является «абстрактным» классом, и служит основой для DOM-узлов. Он обеспечивает базовую функциональность: `parentNode`, `nextSibling`, `childNodes` и т.д. (это геттеры). Объекты класса `Node` никогда не создаются. Но есть определённые классы узлов, которые наследуют от него: `Text` – для текстовых узлов, `Element` – для узлов-элементов и более экзотический `Comment` – для узлов-комментариев.
- Element – это базовый класс для DOM-элементов. Он обеспечивает навигацию на уровне элементов: `nextElementSibling`, `children` и методы поиска: `getElementsByTagName`, `querySelector`. Браузер поддерживает не только HTML, но также XML и SVG. Класс Element служит базой для следующих классов: `SVGElement`, `XMLElement` и `HTMLElement`.
- HTMLElement – является базовым классом для всех остальных HTML-элементов. От него наследуют конкретные элементы:
  - HTMLInputElement – класс для тега `<input>`,
  - HTMLBodyElement – класс для тега `<body>`,

- HTMLAnchorElement – класс для тега `<a>`,
- ...и т.д, каждому тегу соответствует свой класс, который предоставляет определённые свойства и методы.

Таким образом, полный набор свойств и методов данного узла собирается в результате наследования.

Рассмотрим DOM-объект для тега `<input>`. Он принадлежит классу HTMLInputElement.

Он получает свойства и методы из (в порядке наследования):

- `HTMLInputElement` – этот класс предоставляет специфичные для элементов формы свойства,
- `HTMLElement` – предоставляет общие для HTML-элементов методы (и геттеры/сеттеры),
- `Element` – предоставляет типовые методы элемента,
- `Node` – предоставляет общие свойства DOM-узлов,
- `EventTarget` – обеспечивает поддержку событий (поговорим о них дальше),
- ...и, наконец, он наследует от `Object`, поэтому доступны также методы «обычного объекта», такие как `hasOwnProperty`.

Для того, чтобы узнать имя класса DOM-узла, вспомним, что обычно у объекта есть свойство `constructor`. Оно ссылается на конструктор класса, и в свойстве `constructor.name` содержится его имя:

```
alert( document.body.constructor.name ); // HTMLBodyEle
```

...Или мы можем просто привести его к `строке`:

```
alert( document.body ); // [object HTMLBodyElement]
```

Проверить наследование можно также при помощи `instanceof`:

```
alert( document.body instanceof HTMLBodyElement ); // t
alert( document.body instanceof HTMLElement ); // true
alert( document.body instanceof Element ); // true
alert( document.body instanceof Node ); // true
alert( document.body instanceof EventTarget ); // true
```

Как видно, DOM-узлы – это обычные JavaScript объекты. Для наследования они используют классы, основанные на прототипах.

В этом легко убедиться, если вывести в консоли браузера любой элемент через `console.dir(elem)`. Или даже напрямую обратиться к методам, которые хранятся в `HTMLElement.prototype`, `Element.prototype` и т.д.

`console.dir(elem)` и `console.log(elem)`

Большинство браузеров поддерживают в инструментах разработчика две команды: `console.log` и `console.dir`. Они выводят свои аргументы в консоль. Для JavaScript-объектов эти команды обычно выводят одно и то же.

Но для DOM-элементов они работают по-разному:

- `console.log(elem)` выводит элемент в виде DOM-дерева.
- `console.dir(elem)` выводит элемент в виде DOM-объекта, что удобно для анализа его свойств.

Попробуйте сами на `document.body`.

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии
Поделиться
Редактировать на GitHub

Спецификация IDL

В спецификации для описания классов DOM используется не JavaScript, а специальный язык Interface description language (IDL), с которым достаточно легко разобраться.

В IDL все свойства представлены с указанием их типов. Например, `DOMString`, `boolean` и т.д.

Небольшой отрывок IDL с комментариями:

```
// Объявление HTMLInputElement
// Двоеточие ":" после HTMLInputElement означает,
interface HTMLInputElement: HTMLElement {
  // далее идут свойства и методы элемента <input>

  // "DOMString" означает, что значение свойства -
  attribute DOMString accept;
  attribute DOMString alt;
  attribute DOMString autocomplete;
  attribute DOMString value;

  // boolean - значит, что autofocus хранит логиче
  attribute boolean autofocus;
  ...
  // "void" перед методом означает, что данный ме
  void select();
  ...
}
```

Свойство «nodeType»

Свойство `nodeType` предоставляет ещё один, «старомодный» способ узнать «тип» DOM-узла.

Его значением является цифра:

- `elem.nodeType == 1` для узлов-элементов,
- `elem.nodeType == 3` для текстовых узлов,
- `elem.nodeType == 9` для объектов документа,
- В спецификации можно посмотреть остальные значения.

Например:

```
<body>
  <script>
  let elem = document.body;

  // посмотрим что это?
  alert(elem.nodeType); // 1 => элемент

  // и первый потомок...
  alert(elem.firstChild.nodeType); // 3 => текст

  // для объекта document значение типа -- 9
  alert( document.nodeType ); // 9
  </script>
</body>
```

В современных скриптах, чтобы узнать тип узла, мы можем использовать метод `instanceof` и другие способы проверить класс, но иногда `nodeType` проще использовать. Мы не можем изменить значение `nodeType`, только прочитать его.

Тег: nodeName и tagName

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии
Поделиться
Редактировать на GitHub

Получив DOM-узел, мы можем узнать имя его тега из свойств `nodeName` и `tagName`:

Например:

```
alert( document.body.nodeName ); // BODY
alert( document.body.tagName ); // BODY
```

Есть ли какая-то разница между `tagName` и `nodeName`?

Да, она отражена в названиях свойств, но не очевидна.

- Свойство `tagName` есть только у элементов `Element`.
- Свойство `nodeName` определено для любых узлов `Node`:
  - для элементов оно равно `tagName`.
  - для остальных типов узлов (текст, комментарий и т.д.) оно содержит строку с типом узла.

Другими словами, свойство `tagName` есть только у узлов-элементов (поскольку они происходят от класса `Element`), а `nodeName` может что-то сказать о других типах узлов.

Например, сравним `tagName` и `nodeName` на примере объекта `document` и узла-комментария:

```
<body><!-- комментарий -->

  <script>
    // для комментария
    alert( document.body.firstChild.tagName ); // undef
    alert( document.body.firstChild.nodeName ); // #com

    // for document
    alert( document.tagName ); // undefined (не элемент
    alert( document.nodeName ); // #document
  </script>
</body>
```

Если мы имеем дело только с элементами, то можно использовать `tagName` или `nodeName`, нет разницы.

Имена тегов (кроме XHTML) всегда пишутся в верхнем регистре

В браузере существуют два режима обработки документа: HTML и XML. HTML-режим обычно используется для веб-страниц. XML-режим включается, если браузер получает XML-документ с заголовком: `Content-Type: application/xml+xhtml`.

В HTML-режиме значения `tagName/nodeName` всегда записаны в верхнем регистре. Будет выведено `BODY` вне зависимости от того, как записан тег в HTML `<body>` или `<BoDy>`.

В XML-режиме регистр сохраняется «как есть». В настоящее время XML-режим применяется редко.

innerHTML: содержимое элемента

Свойство innerHTML позволяет получить HTML-содержимое элемента в виде строки.

Мы также можем изменять его. Это один из самых мощных способов менять содержимое на странице.

Пример ниже показывает содержимое `document.body`, а затем полностью заменяет его:

```
<body>
  <p>Параграф</p>
```

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии
Поделиться
Редактировать на GitHub

```
  <div>DIV</div>

  <script>
    alert( document.body.innerHTML ); // читаем текущее
    document.body.innerHTML = 'Новый BODY!'; // заменяе
  </script>

</body>
```

Мы можем попробовать вставить некорректный HTML, браузер исправит наши ошибки:

```
<body>

  <script>
    document.body.innerHTML = '<b>тест'; // забыли закр
    alert( document.body.innerHTML ); // <b>тест</b> (и
  </script>

</body>
```

Скрипты не выполнятся

Если `innerHTML` вставляет в документ тег `<script>` – он становится частью HTML, но не запускается.

Будьте внимательны: «innerHTML+=» осуществляет перезапись

Мы можем добавить HTML к элементу, используя `elem.innerHTML+="ещё html"`.

Вот так:

```
chatDiv.innerHTML += "<div>Привет<img src='smile.gif'/>
chatDiv.innerHTML += "Как дела?";
```

На практике этим следует пользоваться с большой осторожностью, так как фактически происходит *не* добавление, а перезапись.

Технически эти две строки делают одно и то же:

```
elem.innerHTML += "...";
// это более короткая запись для:
elem.innerHTML = elem.innerHTML + "..."
```

Другими словами, `innerHTML+=` делает следующее:

1. Старое содержимое удаляется.
2. На его место становится новое значение `innerHTML` (с добавленной строкой).

Так как содержимое «обнуляется» и переписывается заново, все изображения и другие ресурсы будут перезагружены.

В примере `chatDiv` выше строка `chatDiv.innerHTML+="Как дела?"` заново создаёт содержимое HTML и перезагружает `smile.gif` (надеемся, картинка закеширована). Если в `chatDiv` много текста и изображений, то эта перезагрузка будет очень заметна.

Есть и другие побочные эффекты. Например, если существующий текст выделен мышкой, то при переписывании `innerHTML` большинство браузеров снимут выделение. А если это поле ввода `<input>` с текстом, введённым пользователем, то текст будет удалён. И т.д.

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии
Поделиться
Редактировать на GitHub

К счастью, есть и другие способы добавить содержимое, не использующие `innerHTML`, которые мы изучим позже.

outerHTML: HTML элемента целиком

Свойство `outerHTML` содержит HTML элемента целиком. Это как `innerHTML` плюс сам элемент.

Посмотрим на пример:

```
<div id="elem">Привет <b>Мир</b></div>

<script>
  alert(elem.outerHTML); // <div id="elem">Привет <b>Ми
</script>
```

Будьте осторожны: в отличие от `innerHTML`, запись в `outerHTML` не изменяет элемент. Вместо этого элемент заменяется целиком во внешнем контексте.

Да, звучит странно, и это действительно необычно, поэтому здесь мы и отмечаем это особо.

Рассмотрим пример:

```
<div>Привет, мир!</div>

<script>
  let div = document.querySelector('div');

  // заменяем div.outerHTML на <p>...</p>
  div.outerHTML = '<p>Новый элемент</p>'; // (*)

  // Содержимое div осталось тем же!
  alert(div.outerHTML); // <div>Привет, мир!</div> (**)
</script>
```

Какая-то магия, да?

В строке `(*)` мы заменили `div` на `<p>Новый элемент</p>`. Во внешнем документе мы видим новое содержимое вместо `<div>`. Но, как видно в строке (`**`), старая переменная `div` осталась прежней!

Это потому, что использование `outerHTML` не изменяет DOM-элемент, а удаляет его из внешнего контекста и вставляет вместо него новый HTML-код.

То есть, при `div.outerHTML=...` произошло следующее:

- `div` был удалён из документа.
- Вместо него был вставлен другой HTML `<p>Новый элемент</p>`.
- В `div` осталось старое значение. Новый HTML не сохранён ни в какой переменной.

Здесь легко сделать ошибку: заменить `div.outerHTML`, а потом продолжить работать с `div`, как будто там новое содержимое. Но это не так. Подобное верно для `innerHTML`, но не для `outerHTML`.

Мы можем писать в `elem.outerHTML`, но надо иметь в виду, что это не меняет элемент, в который мы пишем. Вместо этого создаётся новый HTML на его месте. Мы можем получить ссылки на новые элементы, обратившись к DOM.

nodeValue/data: содержимое текстового узла

Свойство `innerHTML` есть только у узлов-элементов.

У других типов узлов, в частности, у текстовых, есть свои аналоги: свойства `nodeValue` и `data`. Эти свойства очень похожи при использовании, есть лишь небольшие различия в спецификации. Мы будем использовать `data`, потому что оно короче.

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии
Поделиться
Редактировать на GitHub

Прочитаем содержимое текстового узла и комментария:

```
<body>
  Привет
  <!-- Комментарий -->
  <script>
    let text = document.body.firstChild;
    alert(text.data); // Привет

    let comment = text.nextSibling;
    alert(comment.data); // Комментарий
  </script>
</body>
```

Мы можем представить, для чего нам может понадобиться читать или изменять текстовый узел, но комментарии?

Иногда их используют для вставки информации и инструкций шаблонизатора в HTML, как в примере ниже:

```
<!-- if isAdmin -->
  <div>Добро пожаловать, Admin!</div>
<!-- /if -->
```

...Затем JavaScript может прочитать это из свойства `data` и обработать инструкции.

textContent: просто текст

Свойство `textContent` предоставляет доступ к *тексту* внутри элемента за вычетом всех `<тегов>`.

Например:

```
<div id="news">
  <h1>Срочно в номер!</h1>
  <p>Марсиане атаковали человечество!</p>
</div>

<script>
  // Срочно в номер! Марсиане атаковали человечество!
  alert(news.textContent);
</script>
```

Как мы видим, возвращается только текст, как если бы все `<теги>` были вырезаны, но текст в них остался.

На практике редко появляется необходимость читать текст таким образом.

Намного полезнее возможность записывать текст в `textContent`, т.к. позволяет писать текст «безопасным способом».

Представим, что у нас есть произвольная строка, введённая пользователем, и мы хотим показать её.

- С `innerHTML` вставка происходит «как HTML», со всеми HTML-тегами.
- С `textContent` вставка получается «как текст», все символы трактуются буквально.

Сравним два тега div:

```
<div id="elem1"></div>
<div id="elem2"></div>

<script>
  let name = prompt("Введите ваше имя?", "<b>Винни-пух!

  elem1.innerHTML = name;
```

```
  elem2.textContent = name;
</script>
```

1. В первый `<div>` имя приходит «как HTML»: все теги стали именно тегами, поэтому мы видим имя, выделенное жирным шрифтом.
2. Во второй `<div>` имя приходит «как текст», поэтому мы видим `<b>Винни-пух!</b>`.

В большинстве случаев мы рассчитываем получить от пользователя текст и хотим, чтобы он интерпретировался как текст. Мы не хотим, чтобы на сайте появлялся произвольный HTML-код. Присваивание через `textContent` – один из способов от этого защититься.

Свойство «hidden»

Атрибут и DOM-свойство «hidden» указывает на то, видим ли мы элемент или нет.

Мы можем использовать его в HTML или назначать при помощи JavaScript, как в примере ниже:

```
<div>Оба тега DIV внизу невидимы</div>

<div hidden>С атрибутом "hidden"</div>

<div id="elem">С назначенным JavaScript свойством "hidd

<script>
  elem.hidden = true;
</script>
```

Технически, `hidden` работает так же, как `style="display:none"`. Но его применение проще.

Мигающий элемент:

```
<div id="elem">Мигающий элемент</div>

<script>
  setInterval(() => elem.hidden = !elem.hidden, 1000);
</script>
```

Другие свойства

У DOM-элементов есть дополнительные свойства, в частности, зависящие от класса:

- `value` – значение для `<input>`, `<select>` и `<textarea>` (`HTMLInputElement`, `HTMLSelectElement` …).
- `href` – адрес ссылки «href» для `<a href="...">` (`HTMLAnchorElement`).
- `id` – значение атрибута «id» для всех элементов (`HTMLElement`).
- …и многие другие…

Например:

```
<input type="text" id="elem" value="значение">

<script>
  alert(elem.type); // "text"
  alert(elem.id); // "elem"
  alert(elem.value); // значение
</script>
```

Большинство стандартных HTML-атрибутов имеют соответствующее DOM-свойство, и мы можем получить к нему доступ.

Раздел

Документ

Навигация по уроку

Классы DOM-узлов

Свойство «nodeType»

Тег: nodeName и tagName

innerHTML: содержимое элемента

outerHTML: HTML элемента целиком

nodeValue/data: содержимое текстового узла

textContent: просто текст

Свойство «hidden»

Другие свойства

Итого

Задачи (4)

Комментарии

Раздел
Документ
Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого
Задачи (4)
Комментарии
Поделиться
Редактировать на GitHub

Если мы хотим узнать полный список поддерживаемых свойств для данного класса, можно найти их в спецификации. Например, класс `HTMLInputElement` описывается здесь: https://html.spec.whatwg.org/#htmlinputelement.

Если же нам нужно быстро что-либо узнать или нас интересует специфика определённого браузера – мы всегда можем вывести элемент в консоль, используя `console.dir(elem)`, и прочитать все свойства. Или исследовать «свойства DOM» во вкладке Elements браузерных инструментов разработчика.

Итого

Каждый DOM-узел принадлежит определённому классу. Классы формируют иерархию. Весь набор свойств и методов является результатом наследования.

Главные свойства DOM-узла:

`nodeType`

Свойство `nodeType` позволяет узнать тип DOM-узла. Его значение – числовое: `1` для элементов, `3` для текстовых узлов, и т.д. Только для чтения.

`nodeName/tagName`

Для элементов это свойство возвращает название тега (записывается в верхнем регистре, за исключением XML-режима). Для узлов-неэлементов `nodeName` описывает, что это за узел. Только для чтения.

`innerHTML`

Внутреннее HTML-содержимое узла-элемента. Можно изменять.

`outerHTML`

Полный HTML узла-элемента. Запись в `elem.outerHTML` не меняет `elem`. Вместо этого она заменяет его во внешнем контексте.

`nodeValue/data`

Содержимое узла-неэлемента (текст, комментарий). Эти свойства практически одинаковые, обычно мы используем `data`. Можно изменять.

`textContent`

Текст внутри элемента: HTML за вычетом всех `<теров>`. Запись в него помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст. Можно использовать для защиты от вставки произвольного HTML кода.

`hidden`

Когда значение установлено в `true`, делает то же самое, что и CSS `display:none`.

В зависимости от своего класса DOM-узлы имеют и другие свойства. Например у элементов `<input>` (`HTMLInputElement`) есть свойства `value`, `type`, у элементов `<a>` (`HTMLAnchorElement`) есть `href` и т.д. Большинство стандартных HTML-атрибутов имеют соответствующие свойства DOM.

Впрочем, HTML-атрибуты и свойства DOM не всегда одинаковы, мы увидим это в следующей главе.

Задачи

Считаем потомков

важность: 5

У нас есть дерево, структурированное как вложенные списки `ul/li`.

Напишите код, который выведет каждый элемент списка `<li>`:

1. Какой в нём текст (без поддерева) ?

2. Какое число потомков – всех вложенных `<li>` (включая глубоко вложенные) ?

[Демо в новом окне]

[Открыть песочницу для задачи.]

решение

Что содержит свойство nodeType?

важность: 5

Что выведет этот код?

```
<html>

<body>
  <script>
    alert(document.body.lastChild.nodeType);
  </script>
</body>

</html>
```

решение

Тег в комментарии

важность: 3

Что выведет этот код?

```
<script>
  let body = document.body;

  body.innerHTML = "<!--" + body.tagName + "-->";

  alert( body.firstChild.data ); // что выведет?
</script>
```

решение

Где в DOM-иерархии "document"?

важность: 4

Объектом какого класса является `document` ?

Какое место он занимает в DOM-иерархии?

Наследует ли он от `Node` или от `Element` , или может от `HTMLElement` ?

решение

Раздел

Документ

Навигация по уроку

Классы DOM-узлов

Свойство «nodeType»

Тег: nodeName и tagName

innerHTML: содержимое элемента

outerHTML: HTML элемента целиком

nodeValue/data: содержимое текстового узла

textContent: просто текст

Свойство «hidden»

Другие свойства

Итого

Задачи (4)

Комментарии

Комментарии

перед тем как писать...

Раздел
Документ

Навигация по уроку
Классы DOM-узлов
Свойство «nodeType»
Тег: nodeName и tagName
innerHTML: содержимое элемента
outerHTML: HTML элемента целиком
nodeValue/data: содержимое текстового узла
textContent: просто текст
Свойство «hidden»
Другие свойства
Итого

Задачи (4)
Комментарии