34 | HTML小实验：用代码分析HTML标准 | 极客时间

winter 2019-04-11

你好，我是 winter。

前面的课程中，我们已经讲解了大部分的 HTML 标签。

然而，为了突出重点，我们还是会忽略一些标签类型。比如表单类标签和表格类标签，我认为只有少数前端工程师用过，比如我在整个手机淘宝的工作生涯中，一次表格类标签都没有用到，表单类则只用过 input，也只有几次。

那么，剩下的标签我们怎么样去了解它们呢？当然是查阅 HTML 标准。

由于阅读标准有一定门槛，需要了解一些机制，这节课，我为你设计了一个小实验，用 JavaScript 代码去抽取标准中我们需要的信息。

HTML 标准

我们采用 WHATWG 的 living standard 标准，我们先来看看标准是如何描述一个标签的，这里我们看到，有下面这些内容。

> Categories:
>
> Flow content.
>
> Phrasing content.
>
> Embedded content.
>
> If the element has a controls attribute: Interactive content.
>
> Palpable content.
>
> Contexts in which this element can be used:
>
> Where embedded content is expected.
>
> Content model:
>
> If the element has a src attribute: zero or more track elements, then transparent, but with no media element descendants.
>
> If the element does not have a src attribute: zero or more source elements, then zero or more track elements, then transparent, but with no media element descendants.
>
> Tag omission in text/html:
>
> Neither tag is omissible.
>
> Content attributes:
>
> Global attributes

src — Address of the resource

crossorigin — How the element handles crossorigin requests

poster — Poster frame to show prior to video playback

preload — Hints how much buffering the media resource will likely need

autoplay — Hint that the media resource can be started automatically when the page is loaded

playsinline — Encourage the user agent to display video content within the element's playback area

loop — Whether to loop the media resource

muted — Whether to mute the media resource by default

controls — Show user agent controls

width — Horizontal dimension

height — Vertical dimension

DOM interface:

```
[Exposed=Window, HTMLConstructor]
interface HTMLVideoElement : HTMLMediaElement {
  [CEReactions] attribute unsigned long width;
  [CEReactions] attribute unsigned long height;
  readonly attribute unsigned long videoWidth;
  readonly attribute unsigned long videoHeight;
  [CEReactions] attribute USVString poster;
  [CEReactions] attribute boolean playsInline;
```

```
    };

```

□复制代码

我们看到，这里的描述分为 6 个部分，有下面这些内容。

- Categories：标签所属的分类。
- Contexts in which this element can be used：标签能够用在哪里。
- Content model：标签的内容模型。
- Tag omission in text/html：标签是否可以省略。
- Content attributes：内容属性。
- DOM interface：用 WebIDL 定义的元素类型接口。

这一节课，我们关注一下 Categories、Contexts in which this element can be used、Content model 这几个部分。我会带你从标准中抓取数据，做一个小工具，用来检查 X 标签是否能放入 Y 标签内。

代码角度分析 HTML 标准

HTML 标准描述用词非常的严谨，这给我们抓取数据带来了巨大的方便，首先，我们打开单页面版 HTML 标准 https://html.spec.whatwg.org/

在这个页面上，我们执行一下以下代码：

```
Array.prototype.map.call(document.querySelectorAll(".element"), e=>e.innerText);
```

□复制代码

这样我们就得到了所有元素的定义了，现在有 107 个元素。

不过，比较尴尬的是，这些文本中并不包含元素名，我们只好从 id 属性中获取，最后代码类似这样：

```
var elementDefinations = Array.prototype.map.call(document.querySelectorAll(".element"), e => ({
  text:e.innerText,
```

```
  name:e.childNodes[0].childNodes[0].id.match(/the\-([\s\S]+)\-element:/)?RegExp.$1:null}));
□复制代码
```

接下来我们用代码理解一下这些文本。首先我们来分析一下这些文本，它分成了 6 个部分，而且顺序非常固定，这样，我们可以用 JavaScript 的正则表达式匹配来拆分六个字段。

我们这个小实验的目标是计算元素之间的包含关系，因此，我们先关心一下 categories 和 contentModel 两个字段。

```
for(let defination of elementDefinations) {



  console.log(defination.name + ":")

  let categories = defination.text.match(/Categories:\n([\s\S]+)\nContexts in which this element can be used:/)[1].split("\n");

  for(let category of categories) {

      console.log(category);

  }




/*

  let contentModel = defination.text.match(/Content model:\n([\s\S]+)\nTag omission in text\/html:/)[1].split("\n");

  for(let line of contentModel)

    console.log(line);

*/

}
```

□复制代码

接下来我们来处理 category。

首先 category 的写法中，最基本的就是直接描述了 category 的句子，我们把这些不带任何条件的 category 先保存起来，然后打印出来其它的描述看看：

```
for(let defination of elementDefinations) {


  //console.log(defination.name + ":")
  let categories = defination.text.match(/Categories:\n([\s\S]+)\nContexts in which this element can be used:/)[1].split("\n");
  defination.categories = [];
  for(let category of categories) {
    if(category.match(/^([^ ]+) content./))
      defination.categories.push(RegExp.$1);
    else
      console.log(category)
  }



/*
  let contentModel = defination.text.match(/Content model:\n([\s\S]+)\nTag omission in text\/html:/)[1].split("\n");
```

```
  for(let line of contentModel)
    console.log(line);
*/
}
```

□复制代码

这里我们要处理的第一个逻辑是带 if 的情况。

然后我们来看看剩下的情况：

```
 None.
 Sectioning root.
 None.
 Sectioning root.
 None.
 Form-associated element.
 Listed and submittable form-associated element.
 None.
 Sectioning root.
 None.
 If the type attribute is not in the Hidden state: Listed, labelable, submittable, resettable, and autocapitalize-inheriting form-associated element.
```

```
If the type attribute is in the Hidden state: Listed, submittable, resettable, and autocapitalize-inheriting form-associated element.

Listed, labelable, submittable, and autocapitalize-inheriting form-associated element.

Listed, labelable, submittable, resettable, and autocapitalize-inheriting form-associated element.

None.

Listed, labelable, submittable, resettable, and autocapitalize-inheriting form-associated element.

Listed, labelable, resettable, and autocapitalize-inheriting form-associated element.

Labelable element.

Sectioning root.

Listed and autocapitalize-inheriting form-associated element.

None.

Sectioning root.

None.

Sectioning root.

Script-supporting element.
```


这里出现了几个概念:

- None
- Sectioning root
- Form-associated element
- Labelable element

- Script-supporting element

如果我们要真正完美地实现元素分类，就必须要在代码中加入正则表达式来解析这些规则，这里作为今天的课后问题，留给你自己完成。

接下来我们看看 Content Model，我们照例先处理掉最简单点的部分，就是带分类的内容模型：

```
for(let defination of elementDefinations) {


  //console.log(defination.name + ":")
  let categories = defination.text.match(/Categories:\n([\s\S]+)\nContexts in which this element can be used:/)[1].split("\n");
  defination.contentModel = [];
  let contentModel = defination.text.match(/Content model:\n([\s\S]+)\nTag omission in text\/html:/)[1].split("\n");
  for(let line of contentModel)
    if(line.match(/^([^ ]+) content./))
      defination.contentModel.push(RegExp.$1);
    else
      console.log(line)
}
```

□复制代码

好了，我们照例看看剩下了什么：

> A head element followed by a body element.
>
> If the document is an iframe srcdoc document or if title information is available from a higher-level protocol: Zero or more elements of metadata content, of which no more than one is a title element and no more than one is a base element.
>
> Otherwise: One or more elements of metadata content, of which exactly one is a title element and no more than one is a base element.
>
> Text that is not inter-element whitespace.
>
> Nothing.
>
> Text that gives a conformant style sheet.
>
> One or more h1, h2, h3, h4, h5, h6 elements, optionally intermixed with script-supporting elements.
>
> Nothing.
>
> Zero or more li and script-supporting elements.
>
> Either: Zero or more groups each consisting of one or more dt elements followed by one or more dd elements, optionally intermixed with script-supporting elements.
>
> Or: One or more div elements, optionally intermixed with script-supporting elements.
>
> Either: one figcaption element followed by flow content.
>
> Or: flow content followed by one figcaption element.
>
> Or: flow content.
>
> If the element is a child of a dl element: one or more dt elements followed by one or more dd elements, optionally intermixed with script-supporting elements.
>
> If the element is not a child of a dl element: flow content.

Transparent, but there must be no interactive content or a element descendants.

See prose.

Text.

If the element has a datetime attribute: Phrasing content.

Otherwise: Text, but must match requirements described in prose below.

Nothing.

Transparent.

Zero or more source elements, followed by one img element, optionally intermixed with script-supporting elements.

Nothing.

Zero or more param elements, then, transparent.

Nothing.

If the element has a src attribute: zero or more track elements, then transparent, but with no media element descendants.

If the element does not have a src attribute: zero or more source elements, then zero or more track elements, then transparent, but with no media element descendants.

If the element has a src attribute: zero or more track elements, then transparent, but with no media element descendants.

If the element does not have a src attribute: zero or more source elements, then zero or more track elements, then transparent, but with no media element descendants.

Nothing.

Transparent.

Nothing.

In this order: optionally a caption element, followed by zero or more colgroup elements, followed optionally by a thead element, followed

by either zero or more tbody elements or one or more tr elements, followed optionally by a tfoot element, optionally intermixed with one or more script-supporting elements.

If the span attribute is present: Nothing.

If the span attribute is absent: Zero or more col and template elements.

Nothing.

Zero or more tr and script-supporting elements.

Zero or more td, th, and script-supporting elements.

Nothing.

Zero or more option, optgroup, and script-supporting elements.

Either: phrasing content.

Or: Zero or more option and script-supporting elements.

Zero or more option and script-supporting elements.

If the element has a label attribute and a value attribute: Nothing.

If the element has a label attribute but no value attribute: Text.

If the element has no label attribute and is not a child of a datalist element: Text that is not inter-element whitespace.

If the element has no label attribute and is a child of a datalist element: Text.

Text.

Optionally a legend element, followed by flow content.

One summary element followed by flow content.

Either: phrasing content.

```
Or: one element of heading content.

If there is no src attribute, depends on the value of the type attribute, but must match script content restrictions.

If there is a src attribute, the element must be either empty or contain only script documentation that also matches script content restrictions.

When scripting is disabled, in a head element: in any order, zero or more link elements, zero or more style elements, and zero or more meta elements.

When scripting is disabled, not in a head element: transparent, but there must be no noscript element descendants.

Otherwise: text that conforms to the requirements given in the prose.

Nothing (for clarification, see example).

Transparent

Transparent, but with no interactive content descendants except for a elements, img elements with usemap attributes, button elements, input elements whose type attribute are in the Checkbox or Radio Button states, input elements that are buttons, select elements with a multiple attribute or a display size greater than 1, and elements that would not be interactive content except for having the tabindex attribute specified.
```

□复制代码

这有点复杂，我们还是把它做一些分类，首先我们过滤掉带 If 的情况、Text 和 Transparent。

```
for(let defination of elementDefinations) {

  //console.log(defination.name + ":")

  let categories = defination.text.match(/Categories:\n([\s\S]+)\nContexts in which this element can be used:/)[1].split("\n");

  defination.contentModel = [];
```

```
  let contentModel = defination.text.match(/Content model:\n([\s\S]+)\nTag omission in text\/html:/)[1].split("\n");
  for(let line of contentModel)
    if(line.match(/([^ ]+) content./))
      defination.contentModel.push(RegExp.$1);
    else if(line.match(/Nothing.|Transparent./));
    else if(line.match(/^Text[\s\S]*.$/));
    else
      console.log(line)
}
```

□复制代码

这时候我们再来执行看看：

```
A head element followed by a body element.
One or more h1, h2, h3, h4, h5, h6 elements, optionally intermixed with script-supporting elements.
Zero or more li and script-supporting elements.
Either: Zero or more groups each consisting of one or more dt elements followed by one or more dd elements, optionally intermixed with script-supporting elements.
Or: One or more div elements, optionally intermixed with script-supporting elements.
If the element is a child of a dl element: one or more dt elements followed by one or more dd elements, optionally intermixed with script-supporting elements.
```

See prose.

Otherwise: Text, but must match requirements described in prose below.

Zero or more source elements, followed by one img element, optionally intermixed with script-supporting elements.

Zero or more param elements, then, transparent.

If the element has a src attribute: zero or more track elements, then transparent, but with no media element descendants.

If the element does not have a src attribute: zero or more source elements, then zero or more track elements, then transparent, but with no media element descendants.

If the element has a src attribute: zero or more track elements, then transparent, but with no media element descendants.

If the element does not have a src attribute: zero or more source elements, then zero or more track elements, then transparent, but with no media element descendants.

In this order: optionally a caption element, followed by zero or more colgroup elements, followed optionally by a thead element, followed by either zero or more tbody elements or one or more tr elements, followed optionally by a tfoot element, optionally intermixed with one or more script-supporting elements.

If the span attribute is absent: Zero or more col and template elements.

Zero or more tr and script-supporting elements.

Zero or more td, th, and script-supporting elements.

Zero or more option, optgroup, and script-supporting elements.

Or: Zero or more option and script-supporting elements.

Zero or more option and script-supporting elements.

If the element has a label attribute but no value attribute: Text.

If the element has no label attribute and is not a child of a datalist element: Text that is not inter-element whitespace.

```
If the element has no label attribute and is a child of a datalist element: Text.

When scripting is disabled, in a head element: in any order, zero or more link elements, zero or more style elements, and zero or more meta elements.

When scripting is disabled, not in a head element: transparent, but there must be no noscript element descendants.

Otherwise: text that conforms to the requirements given in the prose.
```

□复制代码

这下剩余的就少多了，我们可以看到，基本上剩下的都是直接描述可用的元素了，如果你愿意，还可以用代码进一步解析，不过如果是我的话，会选择手工把它们写成 JSON 了，毕竟只有三十多行文本。

好了，有了 contentModel 和 category，我们要检查某一元素是否可以作为另一元素的子元素，就可以判断一下两边是否匹配啦，首先，我们要做个索引：

```
var dictionary = Object.create(null);



for(let defination of elementDefinations) {

  dictionary[defination.name] = defination;

}
```

□复制代码

然后我们编写一下我们的 check 函数：

```
function check(parent, child) {
```

```
  for(let category of child.categories)
    if(parent.contentModel.categories.conatains(category))
      return true;
  if(parent.contentModel.names.conatains(child.name))
      return true;
  return false;
}
```

□复制代码

总结

这一节课，我们完成了一个小实验：利用工具分析 Web 标准文本，来获得元素的信息。

通过这个实验，我希望能够传递一种思路，代码能够帮助我们从 Web 标准中挖掘出来很多想要的信息，编写代码的过程，也是更深入理解标准的契机。

我们前面的课程中把元素分成了几类来讲解，但是这些分类只能大概地覆盖所有的标签，我设置课程的目标也是讲解标签背后的知识，而非每一种标签的细节。具体每一种标签的属性和细节，可以留给大家自己去整理。

这一节课的产出，则是“绝对完整的标签列表”，也是我学习和阅读标准的小技巧，通过代码我们可以从不同的侧面分析标准的内容，挖掘需要注意的点，这是一种非常好的学习方法。

bd2star

- □
  2019-04-12
-

被雨水过滤的空气
学习了

- 会飞的大猫
  Winter，刚看完文章，就在淘宝技术节视频看到了你持相机和大家自拍的图片

2019-04-11