

## 11-浏览器：一个浏览器是如何工作的？（阶段二）

你好，我是winter，今天我们继续来看浏览器的相关内容。

我在上一篇文章中，简要介绍了浏览器的工作大致可以分为6个阶段，我们昨天讲完了第一个阶段，也就是通讯的部分：浏览器使用HTTP协议或者HTTPS协议，向服务端请求页面的过程。

今天我们主要来看两个过程：如何解析请求回来的HTML代码，DOM树又是如何构建的。



### 解析代码

我们在前面讲到了HTTP的构成，但是我们有一部分没有详细讲解，那就是Response的body部分，这正是因为HTTP的Response的body，就要交给我们今天学习的内容去处理了。

HTML的结构不算太复杂，我们日常开发需要的90%的“词”（指编译原理的术语token，表示最小的有意义的单元），种类大约只有标签开始、属性、标签结束、注释、CDATA节点几种。

实际上有点麻烦的是，由于HTML跟SGML的千丝万缕的联系，我们需要做不少容错处理。“<?”和“<%”什么的也是必须要支持好的，报了错也不能吭声。

#### 1.词（token）是如何被拆分的

首先我们来看看一个非常标准的标签，会被如何拆分：

```
<p class="a">text text text</p>
```

如果我们从最小有意义单元的定义来拆分，第一个词（token）是什么呢？显然，作为一个词（token），整个p标签肯定是过大了（它甚至可以嵌套）。

那么，只用p标签的开头是不是合适吗？我们考虑到起始标签也是会包含属性的，最小的意义单元其实是“<p”，所以“<p”就是我们的第一个词（token）。

我们继续拆分，可以把这段代码依次拆成词（token）：

- <p “标签开始”的开始；

- class= “a” 属性；
- > “标签开始”的结束；
- text text text 文本；
- </p>标签结束。

这是一段最简单的例子，类似的还有什么呢？现在我们可以来来看看这些词（token）长成啥样子：

示例词	解释
<abc	“开始标签”的开始
a="xxx"	属性
/>	“开始标签”的结束
</xxx>	结束标签
hello world!	文本节点
<!-- xxx -->	注释
<![CDATA[hello world!]]>	CDATA数据节点

根据这样的分析，现在我们讲讲浏览器是如何用代码实现，我们设想，代码开始从HTTP协议收到的字符流读取字符。

在接受第一个字符之前，我们完全无法判断这是哪一个词（token），不过，随着我们接受的字符越来越多，拼出其他的内容可能性就越来越少。

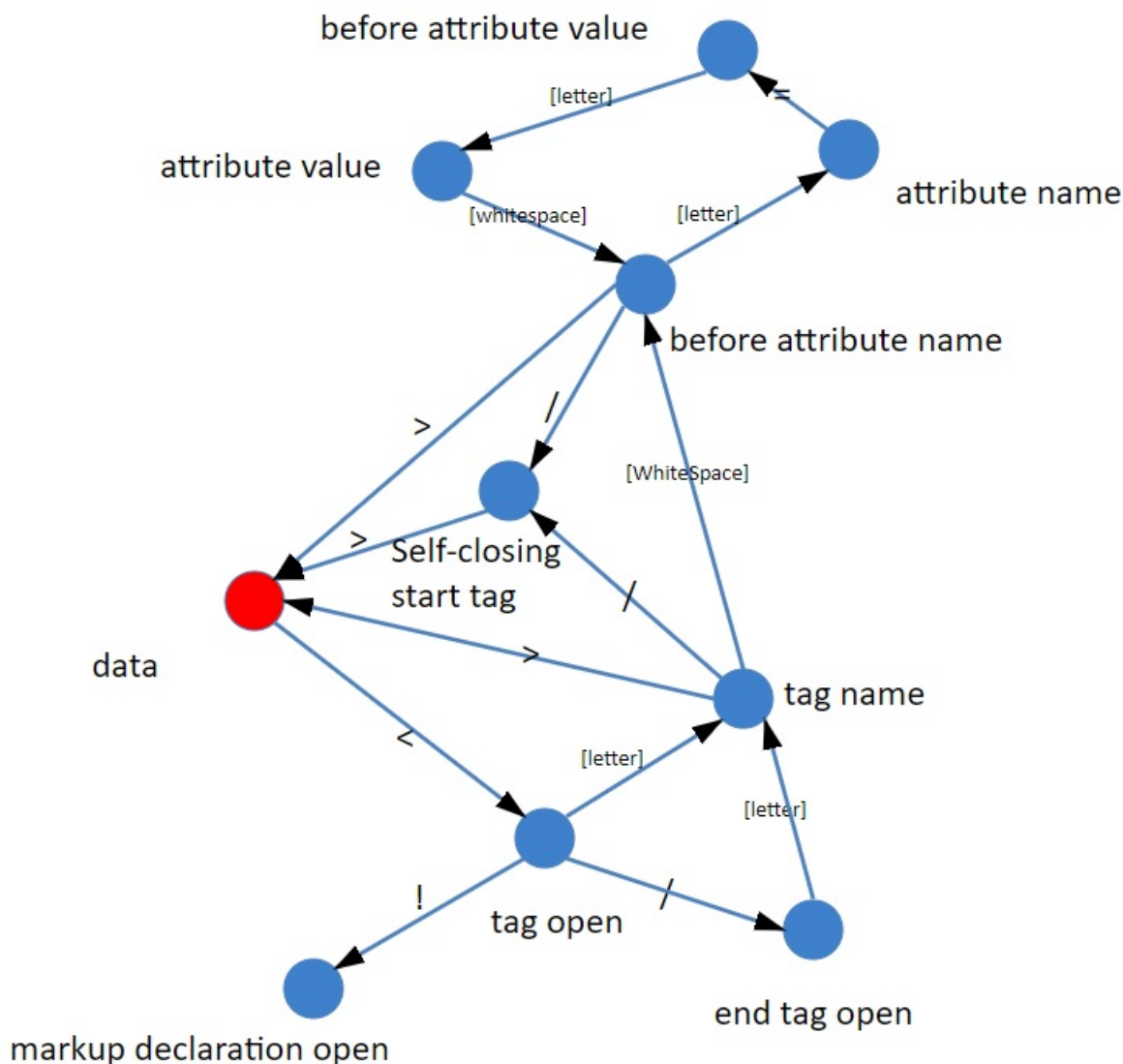
比如，假设我们接受了一个字符“<”我们一下子就知道这不是一个文本节点啦。

之后我们再读一个字符，比如就是x，那么我们一下子就知道这不是注释和CDATA了，接下来我们就一直读，直到遇到“>”或者空格，这样就得到了一个完整的词（token）了。

实际上，我们每读入一个字符，其实都要做一次决策，而且这些决定是跟“当前状态”有关的。在这样的条件下，浏览器工程师要想实现把字符流解析成词（token），最常见的方案就是使用状态机。

## 2.状态机

绝大多数语言的词法部分都是用状态机实现的。那么我们来把部分词（token）的解析画成一个状态机看看：



当然了，我们这里的分析比较粗略，真正完整的HTML词法状态机，比我们描述的要复杂的多。更详细的内容，你可以参考[HTML官方文档](#)，HTML官方文档规定了80个状态（顺便一说，HTML是我见过唯一一个标准中规定了状态机实现的语言，对大部分语言来说，状态机是一种实现而非定义）。

这里我们为了理解原理，用这个简单的状态机就足够说明问题了。

状态机的初始状态，我们仅仅区分 “<” 和 “非<”：

- 如果获得的是一个非<字符，那么可以认为进入了一个文本节点；
- 如果获得的是一个<字符，那么进入一个标签状态。

不过当我们在标签状态时，则会面临着一些可能性。

- 比如下一个字符是 “!”，那么很可能是进入了注释节点或者CDATA节点。
- 如果下一个字符是 “/”，那么可以确定进入了一个结束标签。
- 如果下一个字符是字母，那么可以确定进入了一个开始标签。
- 如果我们要完整处理各种HTML标准中定义的东西，那么还要考虑 “?” “%” 等内容。

我们可以看到，用状态机做词法分析，其实正是把每个词的“特征字符”逐个拆开成独立状态，然后再把所

有词的特征字符链合并起来，形成一个联通图结构。

由于状态机设计属于编译原理的基本知识，这里我们仅作一个简要的介绍。

接下来就是代码实现的事情了，在C/C++和JavaScript中，实现状态机的方式大同小异：我们把每个函数当做一个状态，参数是接受的字符，返回值是下一个状态函数。（这里我希望再次强调下，状态机真的是一种没有办法封装的东西，所以我们永远不要试图封装状态机。）

为了方便理解和试验，我们这里用JavaScript来讲解，图上的data状态大概就像下面这样的：

```
var data = function(c){
    if(c=="&") {
        return characterReferenceInData;
    }
    if(c=="<") {
        return tagOpen;
    }
    else if(c=="\0") {
        error();
        emitToken(c);
        return data;
    }
    else if(c==EOF) {
        emitToken(EOF);
        return data;
    }
    else {
        emitToken(c);
        return data;
    }
};
var tagOpenState = function tagOpenState(c){
    if(c=="/") {
        return endTagOpenState;
    }
    if(c.match(/[A-Z]/)) {
        token = new StartTagToken();
        token.name = c.toLowerCase();
        return tagNameState;
    }
    if(c.match(/[a-z]/)) {
        token = new StartTagToken();
        token.name = c;
        return tagNameState;
    }
    if(c=="?") {
        return bogusCommentState;
    }
    else {
        error();
        return dataState;
    }
};
//.....
```

这段代码给出了状态机的两个状态示例：data即为初始状态，tagOpenState是接受了一个“<”字符，来

判断标签类型的状态。

这里的状态机，每一个状态是一个函数，通过“if else”来区分下一个字符做状态迁移。这里所谓的状态迁移，就是当前状态函数返回下一个状态函数。

这样，我们的状态迁移代码非常的简单：

```
var state = data;
var char
while(char = getInput())
    state = state(char);
```

这段代码的关键一句是“state = state(char)”，不论我们用何种方式来读取字符串流，我们都可以通过state来处理输入的字符流，这里用循环是一个示例，真实场景中，可能是来自TCP的输出流。

状态函数通过代码中的 emitToken 函数来输出解析好的token（词），我们只需要覆盖 emitToken，即可指定对解析结果的处理方式。

词法分析器接受字符的方式很简单，就像下面这样：

```
function HTMLLexicalParser(){

    //状态函数们.....
    function data() {
        // .....
    }

    function tagOpen() {
        // .....
    }
    // .....
    var state = data;
    this.receiveInput = function(char) {
        state = state(char);
    }
}
```

至此，我们就把字符流拆成了词（token）了。

## 构建DOM树

接下来我们要把这些简单的词变成DOM树，这个过程我们是使用栈来实现的，任何语言几乎都有栈，为了给你跑着玩，我们还是用JavaScript来实现吧，毕竟JavaScript中的栈只要用数组就好了。

```
function HTMLSyntacticalParser(){
    var stack = [new HTMLDocument];
    this.receiveInput = function(token) {
```

```
    //.....  
  }  
  this.getOutput = function(){  
    return stack[0];  
  }  
}
```

我们这样来设计HTML的语法分析器，receiveInput负责接收词法部分产生的词（token），通常可以由emmitToken来调用。

在接收的同时，即开始构建DOM树，所以我们的主要构建DOM树的算法，就写在receiveInput当中。当接收完所有输入，栈顶就是最后的根节点，我们DOM树的产出，就是这个stack的第一项。

为了构建DOM树，我们需要一个Node类，接下来我们所有的节点都会是这个Node类的实例。

在完全符合标准的浏览器中，不一样的HTML节点对应了不同的Node的子类，我们为了简化，就不完整实现这个继承体系了。我们仅仅把Node分为Element和Text（如果是基于类的OOP的话，我们还需要抽象工厂来创建对象），

```
function Element(){  
  this.childNodes = [];  
}  
function Text(value){  
  this.value = value || "";  
}
```

前面我们的词（token）中，以下两个是需要成对匹配的：

- tag start
- tag end

根据一些编译原理中常见的技巧，我们使用的栈正是用于匹配开始和结束标签的方案。

对于Text节点，我们则需要把相邻的Text节点合并起来，我们的做法是当词（token）入栈时，检查栈顶是否是Text节点，如果是的话就合并Text节点

同样我们来看看直观的解析过程：

```
<html maaa=a >  
  <head>  
    <title>cool</title>  
  </head>  
  <body>  
      
  </body>
```


```
</html>
```

通过这个栈，我们可以构建DOM树：

- 栈顶元素就是当前节点；
- 遇到属性，就添加到当前节点；
- 遇到文本节点，如果当前节点是文本节点，则跟文本节点合并，否则入栈成为当前节点的子节点；
- 遇到注释节点，作为当前节点的子节点；
- 遇到tag start就入栈一个节点，当前节点就是这个节点的父节点；
- 遇到tag end就出栈一个节点（还可以检查是否匹配）。

我在文章里面放了一个视频，你可以点击查看用栈构造DOM树的全过程。


```
<html maaa=a >
  <head>
    <title>cool</title>
  </head>
  <body>
    
  </body>
</html>
```



parse

String: "\n"

```
{ "name": "html", "maaa": "a" }
  "\n "
    { "name": "head" }
      "\n "
        { "name": "title" }
          "cool"
        "\n "
      "\n "
    { "name": "body" }
      "\n "
        { "name": "img", "src": "a" }
      "\n "
    "\n "
```

 Loading

当我们的源代码完全遵循xhtml（这是一种比较严谨的HTML语法）时，这非常简单问题，然而HTML具有很强的容错能力，奥妙在于当tag end跟栈顶的start tag不匹配的时候如何处理。

于是，这又有一个极其复杂的规则，幸好W3C又一次很贴心地把全部规则都整理地很好，我们只要翻译成对应的代码就好了，以下这个网站呈现了全部规则。你可以点击查看。

<http://www.w3.org/html/wg/drafts/html/master/syntax.html#tree-construction>

## 结语

好了，总结一下。在今天的文章中，我带你继续探索了浏览器的工作原理，我们主要研究了解析代码和构建DOM树两个步骤。在解析代码的环节里，我们一起详细地分析了一个词（token）被拆分的过程，并且给出了实现它所需要的一个简单的状态机。

在构建DOM树的环节中，基本思路是使用栈来构建DOM树为了方便你动手实践，我用JavaScript实现了这一过程。

今天给你留的题目是：在语法和词法的代码，我已经给出了大体的结构，请你试着把内容补充完整吧。

 极客时间

# 重学前端

每天10分钟，重构你的前端知识体系

winter 程劭非  
前手机淘宝前端负责人



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

## 精选留言：

- 曼塔特 2019-02-12 08:45:20  
感觉在看编译原理 [29赞]
- 阿成 2019-02-18 17:25:51  
参考了 github 上的一个 gist，才算写出来个能跑起来的...  
<https://github.com/aimergenge/toy-html-parser> [24赞]

作者回复2019-02-19 13:11:21  
嗯 这个超棒 推荐大家一起看看

- Aaaaaaaaayou 2019-02-13 09:04:51  
return tagOpen 是不是应该改为 return tagOpenState [9赞]

作者回复2019-02-14 19:18:43  
对，我改一下

- umaru 2019-02-21 04:53:12  
老师cdata是啥？(・◇・) [7赞]

作者回复2019-03-01 15:03:14  
XML的相关知识，可以看一下，不怎么重要。

- RMX 2019-02-20 16:24:09  
<https://blog.csdn.net/userkang/article/details/80851153>  
之前在看 Webkit 技术揭秘 这本书，记的笔记。结合老师的文章，了解的更深了。谢谢！ [7赞]



- 是零壹呀 2019-02-19 17:05:10

这一节讲的应该是如何实现一个parser吧。

关于状态机这一块，我觉得是不是可以先讲一节正则的知识点呢。

理解了正则，那么大家对状态机的概念就有了更加直观的理解了。 [3赞]

作者回复2019-03-01 14:39:26

一般正则都是状态机实现的，讲正则对理解它底层的状态机毫无意义啊。

当然了，词法分析也可以用正则来实现，我这里没有这么做而已，我写过一个js的词法分析是用正则做的，你可以参考：

<https://github.com/wintercn/JSinJS/blob/master/source/LexicalParser.js>

- Murphy Demon 2019-02-14 17:40:35

老师可否提供一些课外阅读的材料呢？单纯通过这一篇文章，没有接触相关知识的前提下，get到的东西比较少。 [2赞]

作者回复2019-02-14 20:48:15

这一篇主要涉及的是编译原理，不过我讲的比书简单多了，有个感性认识就可以。

- we 2019-02-13 08:36:24

老师 能回答下，或者给个资料补充一下。手机浏览器与电脑浏览器的区别吗？ [2赞]

作者回复2019-02-14 19:14:15

工作原理上，当然没区别了，但是如果你指兼容性，那三天三夜也说不完……

- 阿成 2019-02-12 17:18:27

老师，为什么状态机没办法封装，能详细解释一下吗 [2赞]

- Nirvana 2019-02-12 16:59:42

老师讲的真好，这部分内容虽完全没接触过，但是相信多听几遍，加上自己的查阅应该也能弄清楚。老师如果开新班请尽快推广，这个课听的太值了。 [2赞]

- leslee 2019-02-12 16:31:49

状态机的图没看懂... [2赞]

- 莲 2019-02-12 14:06:27

这是一篇我不是太懂，却不会自责的文章，毕竟已经涉及浏览器解析html的编译原理了 [2赞]

- 【执着】Paranoid 2019-04-08 10:35:00

赞 [1赞]

- 周飞 2019-03-17 16:42:03

做了一个简单的demo <https://github.com/kobefaith/simpleHtmlParse.git> [1赞]

- [已重置] 2019-03-04 14:24:46

<https://github.com/haven2world/HavenStudyRepository/tree/master/geekbang-winter/htmlParser>

\_(;3」∠)\_ 啰里啰嗦写了一大堆，这大概是我用js写过的最面向对象的东西了 [1赞]

- 风吹一个大耳东 2019-02-20 10:16:30

看到状态机就已经获益匪浅了，老师讲的都是我们平时不在意却又是必须懂的东西~ [1赞]

- 王飞 2019-02-19 22:55:16

老师，感觉在可以讲下virtual-dom [1赞]

作者回复2019-03-01 14:47:57

virtual-dom不是浏览器的东西，算是一种应用技巧吧，我觉得它寿命不会特别长。

- 瞧，这个人 2019-02-16 23:31:02

只简单讲了浏览器怎么解析html,并没有讲具体怎么构建dom树，请寒老师不要偷工减料 [1赞]

作者回复2019-02-19 12:58:58

怎么没讲，还有构造的算法和视频呢，不认真到这个地步了么？

- coma 2019-02-12 22:56:18

请问为什么如果使用基于类的面向对象方式，就要使用抽象工厂来创建对象？ [1赞]

作者回复2019-02-14 19:11:27

这块是设计模式的一个小应用了，因为创建对象的过程无法用接口抽象，所以要用抽象工厂，当然JavaScript里面不是特别有必要用抽象工厂，一般浏览器都是用C++编写的，就一定需要抽象工厂。

- soulful 2019-02-12 20:15:07

看来大学重修一遍编译原理还是值得的 [1赞]