浏览器是怎样工 渲染引擎，HTML解析（连载二）

admin 发表于 2012-05-07

17

类别：前端开发

渲染引擎

渲染引擎的职责是……渲染，也就是把请求的内容显示到浏览器屏幕上。

默认情况下渲染引擎可以显示HTML，XML文档以及图片。 通过插件（浏览器扩展）它可以显示其它类型文档。比如使用PDF viewer插件显示PDF文件。我们会在一个专门的章节讨论插件与扩展。在这一节我们将专注渲染引擎的主要用途——显示用CSS格式化的HTML与图片。

各种渲染引擎

我们提到的Firefox, Safari两种浏览器构建于两种渲染引擎之上：Firefox使用Gecko —— Mozilla自家的渲染引擎；Safari 和 Chrome 都使用 Webkit。

Webkit 是一个开源的渲染引擎，它源自Linux平台上的一个引擎，经过Apple公司的修改可以支持Mac与Windows平台。更多信息可以参考： http://webkit.org/ 。

主要流程

渲染引擎开始于从网络层获取请求内容，一般是不超过8 K 的数据块。接下来就是渲染引擎的基本工作流程：

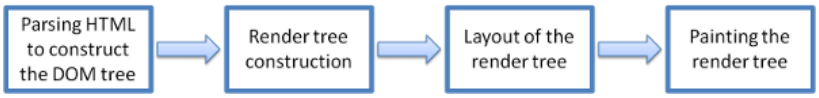

图 2：渲染引擎的基本工作流程（解析ＨＴＭＬ构建ＤOM树，渲染树构建，渲染树布局，绘制渲染树）。
渲染引擎会解析ＨＴＭＬ文档并把标签转换成内容树中的ＤOM节点。它会解析style元素和外部文件中的样式数据。样式数据和ＨＴＭＬ中的显示控制将共同用来创建另一棵树——渲染树。

渲染树包含带有颜色，尺寸等显示属性的矩形。这些矩形的顺序与显示顺序一致。

渲染树构建完成后就是”布局“处理，也就是确定每个节点在屏幕上的确切显示位置。 下一个步骤是 绘制 —— 遍历渲染树并用ＵＩ后端层将每一个节点绘制出来。

一定要理解这是一个缓慢的过程，为了更好的用户体验，渲染引擎会尝试尽快的把内容显示出来。它不会等到所有ＨＴＭＬ都被解析完才创建并布局渲染树。它会 在处理后续内容的同时把处理过的局部内容先展示出来。

主要流程示例

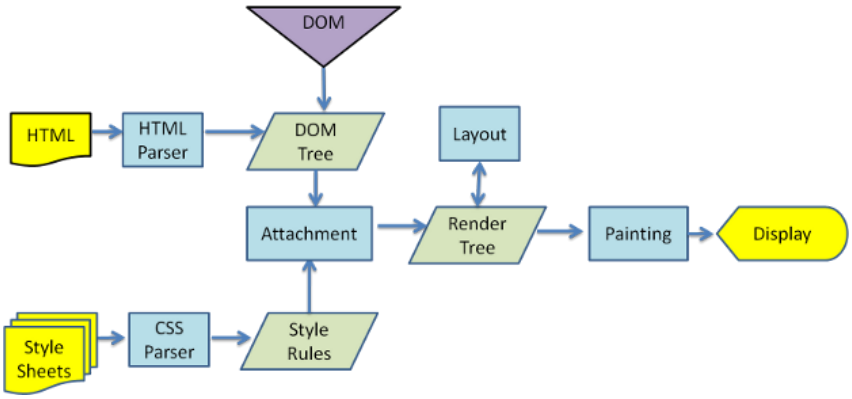

图 3：Webkit主要流程

栏目

前端开发
团队生活
杂谈
活动
用户体验
界面设计
设计规范

标签

CSS Javascript 设计 可用性 交互设计 HTML css3 产品设计 交流会 web 用户体验 表单 翻译 用户研究 Firefox 前端开发 中国象棋 交互 chrome 界面设计 细节 iphone 插件 iPad Canvas ControlJS 杂谈 校验 渲染引擎 浏览器 社会化 旅行 Web标准化 友情提示 iframe html5 js jslint 兼容 pad

兄弟/成员链接

功能

登录
留言板

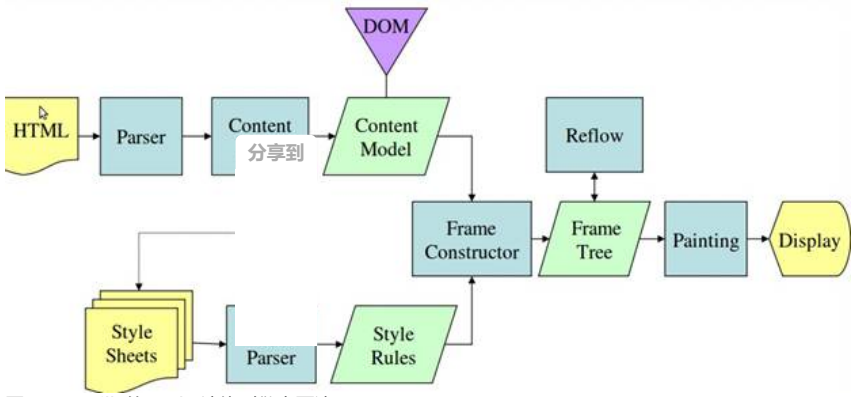

图 4：Mozilla的Gecko渲染引擎主要流程(3.6)
从图3和图4中可以看出，尽管Webkit与Gecko使用略微不同的术语，这个过程还是基本相同的。
Gecko 里把格式化好的可视元素称做“帧树” （Frame tree）。每个元素就是一个帧（frame）。 Webkit 则使用“渲染树”这个术语，渲染树由”渲染对象”组成。Webkit 里使用”layout”表示元素的布局，Gecko则称为”Reflow”。Webkit使用”Attachment”来连接DOM节点与可视化信息以构建渲染树。一个非语义上的小差别是Gecko在HTML与DOM树之间有一个附加的层，称作”content sink”，是创建DOM对象的工厂。我们会讨论流程中的每一部分。

解析

因为解析是渲染引擎中一个很重要的处理，我们会讲的略深入一些。让我们从一个小的解析介绍开始。

解析一个文档意味着把它翻译成有意义的结构以供代码使用。解析的结果通常是一个表征文档的由节点组成的树，称为解析树或句法树。

示例——解析表达式”2 + 3 – 1”可以返回下面的树：

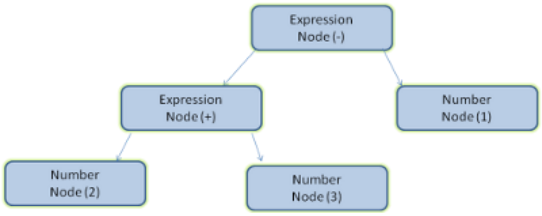

图 5：数学表达式树节点

语法

解析是基于文档所遵循的语法规则——书写所用的语言或格式——来进行的。每一种可以解析的格式必须由确定的语法与词汇组成。这被称之为上下文无关语法。 人类语言并非此种语言，所以不能用常规的解析技术来解析。

解析器——词法分析器组合

解析器有两个处理过程——词法分析与句法分析。

词法分析负责把输入切分成符号序列，符号是语言的词汇——由该语言所有合法的单词组成。

句法分析是对该语言句法法则的应用。

解析器通常把工作分给两个组件——分词程序负责把输入切分成合法符号序列，解析程序负责按照句法规则分析文档结构和构建句法树。词法分析器知道如何过滤像空格，换行之类的无关字符。

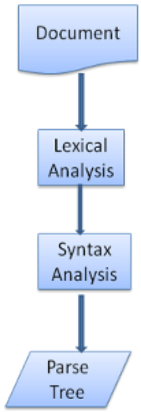

图 6：从源文档到解析树（文档，词法分析，句法分析，解析树）。
解析过程是交互式的。解析器通常会从词法分析器获取新符号并尝试匹配句法规则。如果匹配成功，就在句法树上创建相应的节点，并继续从词法分析器获取下一个符号。如果没有匹配的规则，解析器会内部保存这个符号，并继续从词法分析器获取符号，直到内部保存的所有符号能够成功匹配一个规则。如果最终无法匹配，解析器会抛出异常。这意味着文档无效，含有句法错误。

转换

多数情况下解析树并非最终结果。解析经常是为了从输入文档转换成另外一种格式。比如编译器要把源码编译成机器码，会首先解析成解析树，再把解析树转换成机器码。

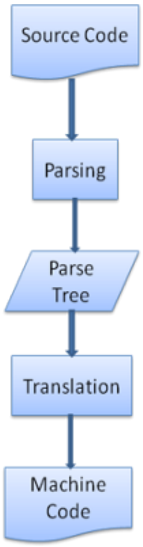

图 7：编译过程（源码，解析，解析树，转换，机器码）。

解析示例

在图5中我们构建了一个数学表达式解析树。让我们来试着定义一个简单的数学语言并看看解析是如何进行的。

词汇：我们的语言可以包含整数，加号和减号。

句法：

- 句法块由表达式，术语及操作符组成。
- 我们的语言可以包含任意数量表达式。
- 表达式定义为术语紧跟着操作符，再跟另外一个术语。
- 操作符是加号或减号。
- 术语可以是整数或表达式。

让我们分析输入”2 + 3 – 1”。

第一个符合规则的子字符串是”2”，根据规则#5它是一个术语。第二个匹配是”2 + 3”，符合第二条规则——一个术语紧跟一个操作符再跟另外一个术语。下一个匹配出现在输入结束时。”2 + 3 – 1”是一个表达式，因为我们已知“2+3”是一个术语，所以符合第二条规则。 “2 + + “不会匹配任何规则，所以是无效的输入。

词法与句法的合法性定义

词汇通常用正则表达式来表示。

比如我们的语言可以定义为:

```
INTEGER :0|[1-9][0-9]*
PLUS : +
MINUS: -
```

如你所见，整型是由正则表达式定义的。

句法常用BNF格式定义，我们的语言被定义为：

```
expression :=  term  operation  term
operation :=  PLUS | MINUS
term := INTEGER | expression
```

我们说过常规解析器只能解析上下文无关语法的语言。这种语言的一个直觉的定义是它的句法可以用BNF完整的表达。其规范定义请参考 http://en.wikipedia.org/wiki/Context-free_grammar

解析器的类型

解析器有两种基本类型——自上而下解析器和自下而上解析器。主观上可以认为自上而下的解析器从上层句法结构开始尝试匹配句法；自下而上的则从输入开始，慢慢转换成句法规则，从底层规则开始，直到上层规则全部匹配。

让我们看看这两种解析器将怎样解析我们的例子：

自上而下解析器从上层规则开始，它会把”2 + 3”定义为表达式，然后定义”2 + 3 – 1”为表达式(定义表达式的过程中也会匹配其它规则，但起点是最高级别规则)。

自下而上的解析器会扫描输入，直到有匹配的规则，它会把输入替换成规则。这样一直到输入结束。部分匹配的规则会放入解析堆栈。

Stack Input

2 + 3 – 1
term + 3 – 1
term operation 3 – 1
expression – 1
expression operation1
expression
这种自下而上的解析器叫作 分享到 解析器，因为输入被向右移动(想象一下一个指针从指向输入开始逐渐向右移动) 并逐渐归约到句法树。

自动创建解析器

有一些工具可以为你创建解 门通常称为解析器生成器。你只需要提供语法——词汇与句法规则——它就能生成一个可以工作的解析器。创建解析器 斤器有深入的了解，并且手动创建一个优化的解析器并不容易，所以解析器生成工具很有用。

Webkit使用两款知名的解析器生成工具：Flex用于创建词法分析器，Bison用于创建解析器 (你也许会看到它们以Lex和Yacc的名字存在)。Flex的输入文件是符号的正则表达式定义，Bison的输入文件是ＢＮＦ格式的句法定义。

HTML解析器

ＨＴＭＬ解析器的工作是解析ＨＴＭＬ标记到解析树。

HTML语法定义

ＨＴＭＬ的词汇与句法定义在w3c组织创建的规范中。当前版本是HTML4，HTML5的工作正在进行中。

不是上下文无关语法

在对解析器的介绍中看到，语法可以用类似ＢＮＦ的格式规范地定义。不幸的是所有常规解析器的讨论都不适用于ＨＴＭＬ（我提及它们并不是为了娱乐，它们可以用于解析CSS和JavaScript）。HTML无法用解析器所需的上下文无关的语法来定义。过去HTML格式规范由DTD (Document Type Definition)来定义，但它不是一个上下文无关语法。

HTML与XML相当接近。XML有许多可用的解析器。HTML还有一个XML变种叫XHTML，那么它们主要区别在哪里呢？区别在于HTML应用更加”宽容”，它容许你漏掉一些开始或结束标签等。它整个是一个“软”句法，不像XML那样严格死板。 总的来说这一看似细微的差别造成了两个不同的世界。一方面这使得HTML很流行，因为它包容你的错误，使网页作者的生活变得轻松。另一方面，它使编写语法格式变得困难。所以综合来说，ＨＴＭＬ解析并不简单，现成的上下文相关解析器搞不定，ＸＭＬ解析器也不行。

HTML DTD

HTML的定义使用DTD文件。这种格式用来定义SGML族语言，它包含对所有允许的元素的定义，包括它们的属性和层级关系。如我们前面所说，HTML DTD构不成上下文无关语法。

DTD有几种不同类型。严格模式完全尊守规范，但其它模式为了向前兼容可能包含对早期浏览器所用标签的支持。当前的严格模式ＤＴＤ：http://www.w3.org/TR/html4/strict.dtd

DOM

解析器输出的树是由ＤＯＭ元素和属性节点组成的。DOM的全称为：Document Object Model。它是ＨＴＭＬ文档的对象化描述，也是ＨＴＭＬ元素与外界（如Javascript）的接口。

ＤＯＭ与标签几乎有着一一对应的关系，如下面的标签

```
<html>
	<body>
		<p>
			Hello World
		</p>
		<div> <img src="example.png"/></div>
	</body>
</html>
```

会被转换成如的ＤＯＭ树：

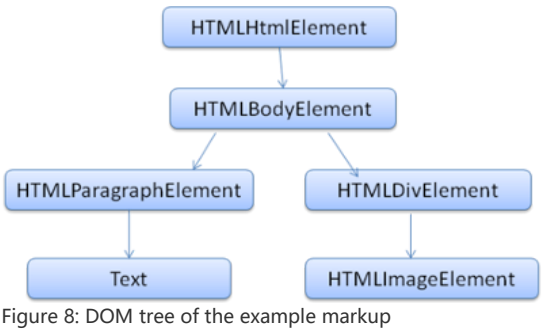

Figure 8: DOM tree of the example markup

与HTML一样，DOM规范也由w3c组织制订。参考：http://www.w3.org/DOM/DOMTR. 这是一个操作文档的通用规范。有一个专门的模块定义HTML特有元素: http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/idl-definitions.html.

当我们说树中包含DOM节点时，意思就是这个树是由实现了DOM接口的元素组成。这些实现包含了其它一些浏览器内部所需的属性。

解析算法

如我们前面看到的，HTML无法使用自上而下或自下而上的解析器来解析。

理由如下：

- 语言的宽容特点
- 浏览器需要对无效HTML 性的事实。
- 解析过程的反复。通常解 源码不会变化。但在HTML中，script标签包含“document.write”时可以添加内容，即解析过程实际上还会改变源

浏览器创建了自己的解析器来解析HTML文档。

HTML5规范里对解析算法有具体的说明，解析由两部分组成：分词与构建树。

分词属于词法分析部分，它把输入解析成符号序列。在HTML中符号就是开始标签，结束标签，属性名称和属生值。

分词器识别这些符号并将其送入树构建者，然后继续分析处理下一个符号，直到输入结束。

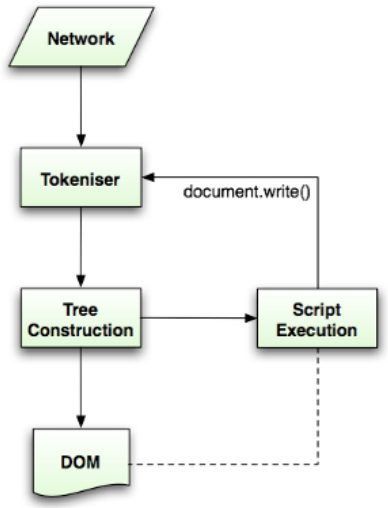

图 6: HTML解析流程 (源自HTML5规范)

分词算法

算法的输出是HTML符号。算法可以用状态机来描述。 每一个状态从输入流中消费一个或多个字符，并根据它们更新下一状态。决策受当前符号状态和树的构建状态影响。这意味着同样的字符可能会产生不同的结果，取决于当前的状态。算法太复杂，我们用一个例子来看看它的原理。

基础示例，分析下面的标签:

```
<html>
        <body>
                Hello world
        </body>
</html>
```

初始状态是”Data state”，当遇到”<“时状态改为**“Tag open state”**。吃掉”a-z”字符组成的符号后产生了”Start tag token”，状态变更为**“Tag name state”**。我们一直保持此状态，直到遇到”>”。每个字符都被追加到新的符号名上。在我们的例子中，解出的符号就是”html”。

当碰到”>”时，当前符号完成，状态改回**“Data state”**。”<body>”标签将会以同样的方式处理。现在”html”与”body”标签都完成了，我们回到**“Data state”**状态。吃掉”H”（”Hello world”第一个字母）时会产生一个字符符号，直到碰到”</body>”的”<“符号，我们就完成了一个字符符号”Hello world”。

现在我们回到**“Tag open state”**状态。吃掉下一个输入”/”时会产生一个”end tag token”并变更为**“Tag name state”**状态。同样，此状态保持到我们碰到”>”时。这时新标签符号完成，我们又回到**“Data state”**。同样”</html>”也会被这样处理。

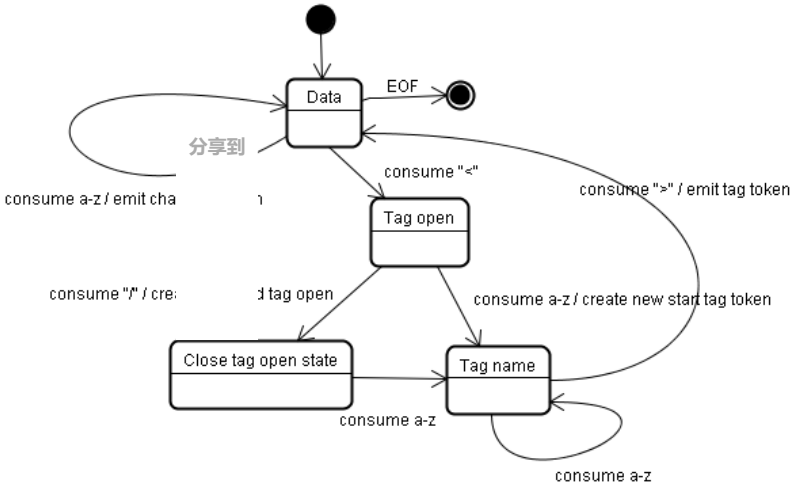

图 9: 示例输入源的分词处理

树的构建算法

当解析器被创建时，文档对象也被创建了。在树的构建过程中DOM树的根节点（Documen）将被修改，元素被添加到上面去。每个分词器完成的节点都会被树构建器处理。规范中定义了每一个符号与哪个DOM对象相关。除了把元素添加到DOM树外，它还会被添加到一个开放元素堆栈。这个堆栈用于纠正嵌套错误和标签未关闭错误。这个算法也用状态机描述，它的状态叫做“insertion modes”。

让我们看看下面输入的树构建过程：

```
<html>
        <body>
                Hello world
        </body>
</html>
```

树的构建过程中，输入就是分词过程中得到的符号序列。第一个模式叫**“initial mode”**。接收 html 符号后会变成**“before html”**模式并重新处理此模式中的符号。这会创建一个HTMLHtmlElement元素并追加到根文档节点。

然后状态改变为**“before head”**。我们收到”body”时，会隐式创建一个HTMLHeadElement，尽管我们没有这个标签，它也会被创建并添加到树中。

现在我们进入**“in head”**模式，然后是**“after head”**，Body会被重新处理，创建HTMLBodyElement元素并插入，然后进入**“in body”**模式。

字符符号”Hello world”收到后会创建一个”Text”节点，所有字符都被一一追加到上面。

收到body结束标签后进入 **“after body”** 模式，收到html结束标签后进入**“after after body”**模式。所有符号处理完后将终止解析。

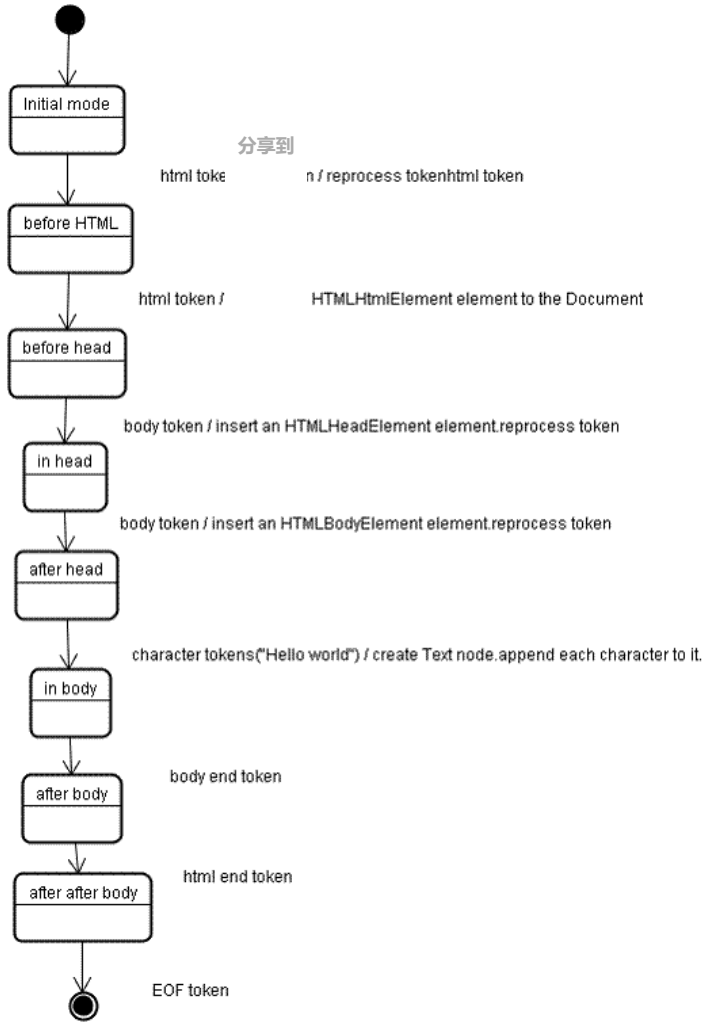

图 10: 示例HTML树的构建

解析结束后的动作

在这一阶段浏览器会把文档标记为交互模式，并开始解析deferred模式的script。” deferred” 意味着脚本应该在文档解析完成后执行。脚本处理完成后将进入” complete” 状态，” load” 事件发生。

HTML5规范中包含了完整的算法： http://www.w3.org/TR/html5/syntax.html#html-parser

浏览器的容错

你永远不会看到HTML页面语法错误。浏览器会修正错误并继续。看看下面的例子：

```
<html>
  <mytag>
  </mytag>
  <div>
  <p>
  </div>
       Really lousy HTML
  </p>
</html>
```

我一定违背了几百万条规则（” my tag” 是非法标签，” p” 与” div” 元素嵌套错误等等），但浏览器仍然正确地显示，没有任何抱怨。所以很多解析器代码在修正这些HTML作者的错误。

浏览器的错误处理相当统一，惊人的是这并不是当前HTML规范的一部分，就像书签、前进、后退，只是多年以来在浏览器中开发出来的。有些无效的HTML结构出现在许多网站，浏览器会尝试用和其它各种浏览器一致的方式修复这些错误。

HTML5规范中应这一需求定义了一些东西，Webkit在它的HTML解析器类开头的注释中很好的做了摘要：

解析器分析输入符号生成文档，并构建文档树。如果文档格式良好，解析工作会很简单。

不幸的是，我们要处理很多格式不良的HTML文档，解析器需要宽容这些错误。

我们至少需要照顾下列错误：

1. 元素必需被插入在正确的位置。未关闭的标签应该一一关闭，直到可以添加新元素。

2. 不允许直接添加元素。用户可能会漏掉一些标签，比如：HTML HEAD BODY TBODY TR TD LI（我遗漏了什么？）。

3. 在inline元素里添加block元素时，应关闭所有inline元素，再添加block元素。

4. 如果以上不起作用，关闭所有元素，直到可以添加，或者忽略此标签。

让我们来看一些Webkit容错的例子：

**使用</br>代替
**

有些站点使用</br>而不是
了更好的与IE和Firefox兼容，Webkit将其视为
。代码如下：

```
if (t->isCloseTag(brTag)          ument->inCompatMode()) {
     reportError(Malform          );
     t->beginTag = true;
}
```

注意，这里的错误处理是内部的，下会显示给用户。

迷失的表格

像下面的例子这样，一个表格包含在另外一个表格的内容中，但不是在外部表格的单元格里：

```
<table>
        <table>
                <tr><td>inner table</td></tr>
         </table>
        <tr><td>outer table</td></tr>
</table>
```

Webkit会改变层级关系，把它们处理成两个相临的表格：

```
<table>
        <tr><td>outer table</td></tr>
</table>
<table>
        <tr><td>inner table</td></tr>
 </table>
```

代码：

```
if (m_inStrayTableContent && localName == tableTag)
        popBlock(tableTag);
```

Webkit用一个堆栈保存当前元素，它会把里面的表格弹出到外部表格堆栈，使它们成为兄弟表格。

元素嵌套

为防止一表单的嵌套，第二个表单会被忽略。代码：

```
if (!m_currentFormElement) {
        m_currentFormElement = new HTMLFormElement(formTag,     m_document);
}
```

过深的元素层级

注释不言自喻：

```
过深的典型，它用大量的<b>嵌套到1500个标签的深度。我们只允许同一标签连续出现20次，超过的话，所有此标签都会被忽略。
```

```
bool HTMLParser::allowNestedRedundantTag(const AtomicString& tagName)
{

unsigned i = 0;
for (HTMLStackElem* curr = m_blockStack;
         i < cMaxRedundantTagDepth && curr && curr->tagName == tagName;
     curr = curr->next, i++) { }
return i != cMaxRedundantTagDepth;
}
```

错误的html或body结束标签位置

注释仍然很明了：

```
支持真正的错误html
我们永远不关闭tag，因为有些愚蠢的网页在文档真正结束之前就关闭了它。
让我们用end()来关闭标签。

if (t->tagName == htmlTag || t->tagName == bodyTag )
        return;
```

所以网页作者们小心了，除非你想写一个Webkit容错的示例代码，否则请按正确格式书写HTML。

本文作者：admin 转载请注明来自：**携程设计委员会**

Tags: HTML 浏览器 渲染引擎

浏览器是怎样工作的（一）：基础知识 LabJs的分析图

分享到

*17*条评论

目前浏览器这么多，渲染效果有很多都各不相同，。。。。。。。。。。。。。。。。。。真应该向小志学习，直接抵制IE内核的留浏览器。。他们早晚得统一规则，没统一之前，我们。。。。。。。。。。。。。。苦啊 ToT #16

书生 回复

图床挂了。不翻看不到图。 #15

我的职业生涯 回复

是篇很不错的文章，详细到乍舌，就是太理论性，可用的干货不多。 #14

葵中剑 回复

@葵中剑 这个就是讲理论的，要是干活多的话，你就可以去开发浏览器引擎了

in 回复

说实话，有些代码不是很清楚什么意思，最好加一个注释哦，呵呵 #13

最科技 回复

笔者的计算机基础一定很牛~ 提到词法和句法分析，让我想到了当年的汇编语言~ 可惜当初没好好学啊~ #12

小灰灰 回复

这个太复杂了 :mrgreen: #11

www.tdwg.net 回复

太复杂了，表示压力好大呀 #10

大漠 回复

🙂 #9

netyc 回复

➡️ 这个是翻译的吧，原文你要贴个链接说明一下啊…而且还没写完 #8

晕死 回复

英文链接希望贴出地址 #7

bird 回复

😥 晦涩难懂啊~ #6

夜之花狸 回复

翻看了好几个网站的UED。发现携程毕竟专注与技术研究 哈哈 #5

找白菜 回复

好详细啊，看不太懂。看来有必要长期研究一下 #4

铁板烧设备厂家 回复

谢谢啊，这个是你原创的吗 #3

フェラガモ 靴 回复

😀 谢谢博主分享质量很高的文章。 #2

新一 回复

图片看不到...

yan 回复 #1

分享到

我来说说

签名: (必须)

邮箱: (必须，为您保密)

网址:

留言:

验证码 *

提交评论 取消 有人回复时邮件通知我