

# WEB前端标准在各浏览器 器中的实现差异

路人甲

2010-08-28



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# 标准都说了什么？

- 文档格式：HTML
- 布局：CSS
- 与宿主环境有关脚本语言：Javascript
- 文档操作方式：DOM



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# 主要话题

- IE 的 hasLayout 与 Block Formatting Content 规范异同
- line box 与 Inline Formatting Contents 的理解
- DOM Attributes 与 JS Object 差异
- tracemonkey的脚本优化
- HTTP头内编码声明在浏览器内的容错影响



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# IE HAS LAYOUT AND BLOCK FORMATTING CONTENTS



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# IE 的 haslayout 与 Block Formatting Contents 规范异同

**'Layout' 是 IE 的专有概念**，它决定了元素如何对其内容进行定位和尺寸计算，与其他元素的关系和相互作用，以及对应用还有使用者的影响。

MSDN **"HasLayout" Overview** :

[http://msdn.microsoft.com/en-us/library/bb250481\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb250481(VS.85).aspx)

**Block Formatting Contexts (块格式化上下文)** 是 W3C CSS 2.1 规范中的一个概念，它决定了元素如何对其内容进行定位，以及与其他元素的关系和相互作用。

在 BFC 中，每一个元素左外边与包含块的左边相接触（对于从右到左的格式化，右外边接触右边），**即使存在浮动也是如此**（尽管一个元素的内容区域会由于浮动而压缩），除非这个元素也创建了一个新的 BFC。

浮动元素、绝对定位元素、inline-blocks、table-cells、table-captions 以及 'overflow' 值不是 'visible' 的元素，会创建 BFC。

【注】：CSS3 中，将 Block formatting contexts 叫做 flow root。对于触发方式也做了修改：The value of 'position' is neither 'static' nor 'relative'

见 [CSS3]： <http://www.w3.org/TR/css3-box/#block-level0>



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 WebAdvocate 很彪悍



# IE 的 hasLayout 与 Block Formatting Contents 规范异同

包含浮动元素特性 ( 浮动清理 ) :

```
<div style="width:300px; background:gray;">  
  <div style="float:left; background:gold;">float:left</div>  
</div>
```

```
<div style="width:300px; background:gray; overflow:auto;">  
  <div style="float:left; background:gold;">float:left</div>  
</div>
```

float:left

在触发 hasLayout 的元素和创建了 Block Formatting Contexts 的元素中，浮动元素参与高度的计算。

【注】:IE8 标准文档模式下触发了 haslayout 特性的元素不再拥有"清理" 浮动元素功能，它回归了标准的 Block Formatting Contexts 规范，只有创建 BFC 后才拥有此功能。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# IE 的 hasLayout 与 Block Formatting Contents 规范异同

不能与浮动元素相互覆盖特性：

```
<div style="width:300px; height:150px; background:gray">  
  <div style="float:left;background:gold;margin:0 10px;" >  
    Float Block  
  </div>  
  <div style="background:green; zoom:1;">  
    Block Formatting Content/haslayout  
  </div>  
</div>
```

```
<div style="width:300px; height:150px; background:gray">  
  <div style="float:left;background:gold;margin:0 10px;" >  
    Float Block  
  </div>  
  <div style="background:green; overflow:auto;">  
    Block Formatting Content/haslayout  
  </div>  
</div>
```



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# IE 的 hasLayout 与 Block Formatting Contents 规范异同

不能与浮动元素相互覆盖特性：

Float Block

If I had a single  
flower for every time I  
think about you, I  
could walk forever in  
my garden.

在触发 hasLayout 的元素和创建了 Block Formatting Contexts 的元素中，浮动元素参与高度的计算。

【注】:IE8 标准文档模式下 zoom:1 不再触发 haslayout 特性。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# IE 的 hasLayout 与 Block Formatting Contents 规范异同

在标准文档模式中，合理利用 **block formatting context** 与 IE 的 **haslayout** 特性实现两列文字布局。

```
<style>
  * {margin:0;padding:0; font-size:14px;}
  ol {list-style:none;width:200px;}
  ol li {overflow:hidden;}
  ol li strong {float:left; }
  ol li p {_display:inline;overflow:auto;zoom:1;} /*用inline hack修复IE6 bug*/
</style>
<ol>
  <li><strong>内容: </strong><p>“地球一小时”</p></li>
  <li><strong>内容.内容: </strong><p>“地球一小时”</p></li>
  <li><strong>内容.容: </strong><p>“地球一小时”</p></li>
  <li><strong>内容.内容.内容: </strong><p>“地球一小时”</p></li>
</ol>
```



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

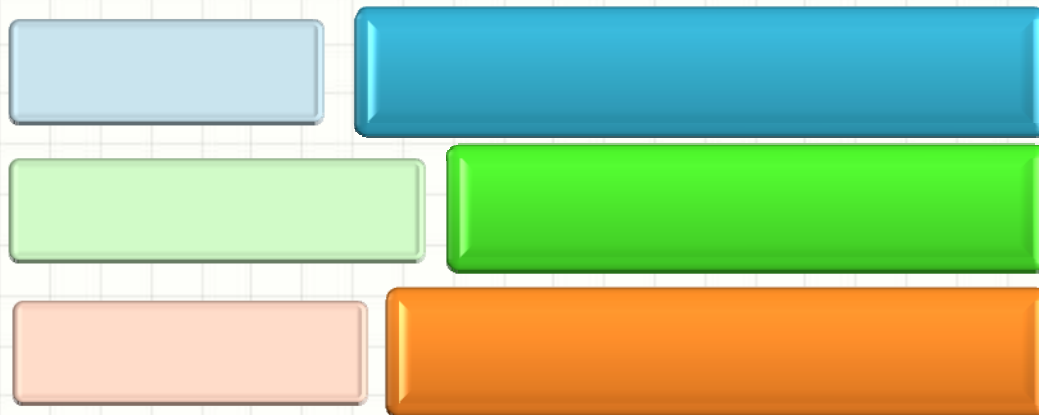
# IE 的 hasLayout 与 Block Formatting Contents 规范异同

第一：做人要厚道

第十一：仍然要厚道

第一百一：厚道是必要的


第二十万三千：.....



利用 BFC 和 hasLayout 清理 LI 容器内浮动元素，并利用 BFC 和 haslayout 区域不与浮动框重合的特性，简单的实现两列文字布局。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# LINE BOX AND INLINE FROMATTING CONTENT



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# Line Box 与 Inline Formatting Content

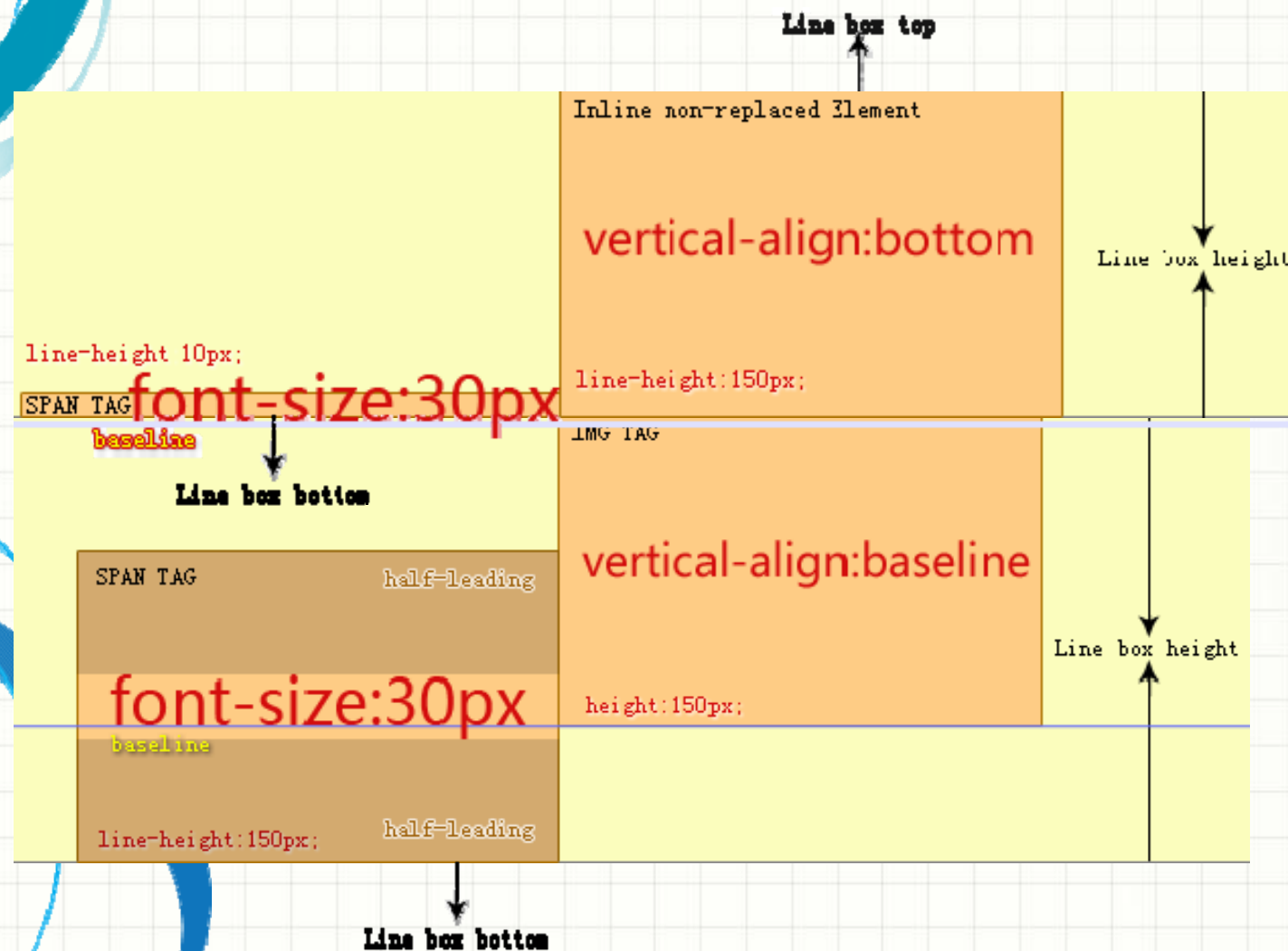
**Line Box 是个计算值，它由 line-height (height) 和 vertical-align 特性值共同决定的，多个 Line Boxes 构成了行内格式化上下文**

- 每一个行内元素会产生一个行内框；
- 行内框会在行框内横向排列；
- 'line-height' 特性值指定了每个行内非替换元素生成的行内框的 **确切** 高度；行内替换元素的高度由 'height' 特性值决定；
- 文字在行内框中垂直排列，上下空隙用半差异填补；如果字号大于行内框则文字从上下方向上溢出行内框，并可能渗入到其他行框内（行框是永远不会重叠的）；
- 'vertical-align' 特性值指定了每个行内框的垂直对齐方式；
- 行框的**顶边界**是这一系列垂直对齐的行内框最高的顶边框，**底边界**是最低的底边框。
- **行框的高度**是顶边界到底边界的距离。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

## 正常 Line Box 计算示意图

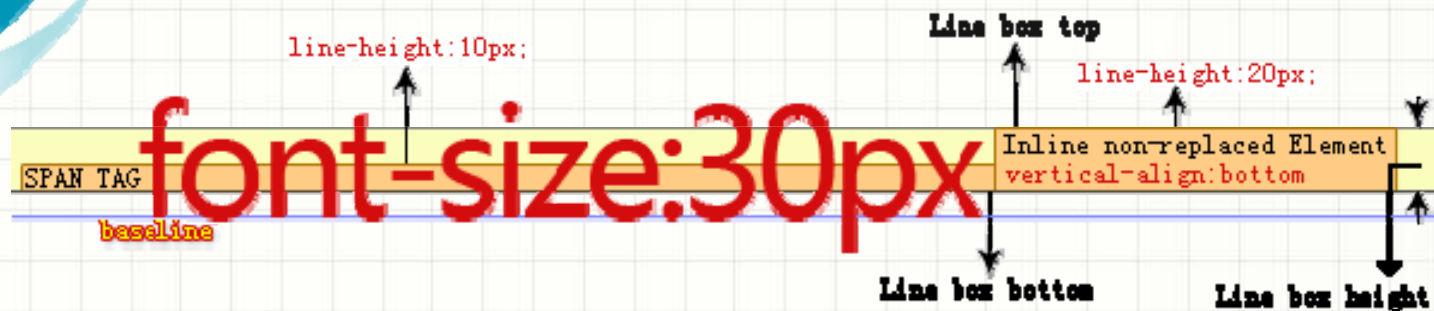


欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



## IE6 IE7 IE8(Q) 下 Line Box 计算错误

正确的计算



错误的计算



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

## IE6 IE7 IE8(Q) 下 Line Box 计算错误

正确的计算



错误的计算



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# DOM ATTRIBUTES AND JAVASCRIPT PROPERTY



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# DOM Attributes

这需从DOM-Level-2的附录D说起：

## Appendix D: ECMAScript Language Binding

根据 W3C DOM 规范附录中 DOM 对象与 ECMAScript 规范中对象的绑定关系可知，DOM 本身是一套接口实现，调用相应的方法或取得相应的属性，其返回值是严格遵循 DOM 本身规范的。而 ECMAScript 规范中描述的对象类型有其自身意义范畴，当使用 ECMAScript 语法调用 DOM 是需要将两者提供的不同对象类型间的绑定。比如某 DOM 方法返回一个 Element 对象，他被绑定到 ECMAScript 中一个对象 (Object) 上，如果对这个 Object 某属性进行赋值和取值操作，究竟是对 Object 对象本身属性还是对 Element 对象属性做相关操作呢？这就产生了二意性。由此 DOM 规范中提供了 `getAttribute` 与 `setAttribute` 方法用来明确为 Element 对象属性做赋值取值操作。



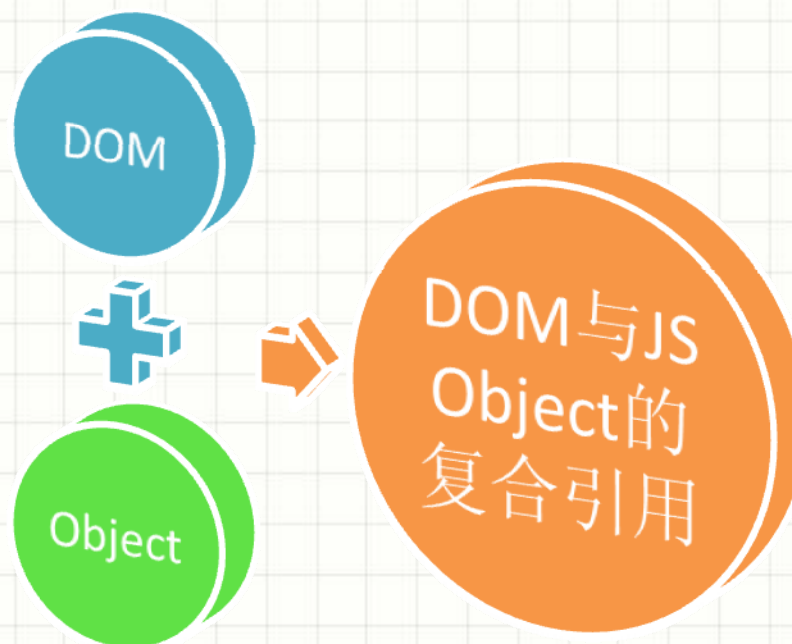
欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

这两者是绑定关系，但是他们都有相应的方法设置、修改和删除属性。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍






HTML ELEMENT =  
HTML的标签属性 ( Element Attribute )  
+ JS 对象内属性 ( Object Property )

<Element Attributes /> + {JS Object: Property}

DOMObject.\*Attribute? 方法用来对 Element 内 String 型属性进行处理  
DOMObject.\* 方法用来对 JS Object 内 property 属性进行处理



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



```
<div id="A"> </div>
<script>
var A = document.getElementById("A");
A.setAttribute("myAttribute","DOMAttribute")
A.myAttribute = "ObjectProperty";
A.innerHTML = A.getAttribute("myAttribute") + "：" + A.myAttribute;
</script>
```

**IE6 IE7 IE8 print :**  
**ObjectProperty : ObjectProperty**

**other print:**  
**DOMAttribute: ObjectProperty**

**IE 8 及以下版本浏览器中混淆了 DOMAttribute 和 ObjectProperty 导致以上问题出现。**



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# TRACEMONKEY SCRIPT OPTIMIZATION



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# tracemonkey 的脚本优化

tracemonkey 就是 Firefox 的脚本引擎啦~~

```
function foo(){return 2+4};  
alert(foo);
```

**Firefox print :**

```
function foo(){return 6}
```

**other print:**

```
function foo(){return 2+4};
```



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# tracemonkey 的条件函数表达式

逻辑语句中的函数定义会被转换为函数表达式，这与它本身的优化机制还有关系。

**先来看对逻辑语句的优化：**

```
function foo(){  
    if ( true ) {  
        function A(){alert('A')}  
    }else{  
        function A(){alert('B')}  
    }  
    A();  
}  
foo();  
alert(foo);
```



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# tracemonkey 的条件函数表达式

Firefox print :

A

```
function foo() {  
  function A() {  
    alert("A");  
  }  
  A();  
}
```

other print:

B

```
function foo(){  
  if ( true ) {  
    function A(){alert('A')}  
  }else{  
    function A(){alert('B')}  
  }  
  A();  
}
```

Firefox 中不可能被运行到的分支被优化掉了。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# tracemonkey 的条件函数表达式

规避其优化策略，可以使用计算值代替 TRUE 。

**使用计算值避免Firefox对逻辑语句的优化：**

```
function foo(){  
    if ( 1 ===1 ) {  
        function A(){alert('A')}  
    }else{  
        function A(){alert('B')}  
    }  
    A();  
}  
foo();  
alert(foo);
```

这样 **Firefox** 就不会优化代码内容了。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# tracemonkey 的条件函数表达式

这就没问题了么？还是看看结果吧：

**Firefox print:**

**A**

```
function foo(){
  if ( 1===1 ) {
    function A(){alert('A')}
  }else{
    function A(){alert('B')}
  }
  A();
}
```

**other print:**

**B**

```
function foo(){
  if ( 1===1 ) {
    function A(){alert('A')}
  }else{
    function A(){alert('B')}
  }
  A();
}
```

!! ?? !! 无语了 ==b



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# tracemonkey 的条件函数表达式

实际上问题是这样的：

JS 引擎在处理脚本程序之前会对内容作语法整理，由于 JS 内没有块级作用域，函数声明会被提高到当前执行块 (Global 或 Function) 的顶端执行。

语法树整理后代码：

```
function foo(){  
    function A(){alert('A')};  
    function A(){alert('B')};  
    if ( 1===1 ) {}else{}  
    A();  
}
```

同名函数A覆盖前面的函数A，导致执行结果为B。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# tracemonkey 的条件函数表达式

tracemonkey 则处理的不是这么"规范"。这个引擎太聪明了，它认为条件语句内的函数声明应该是函数表达式，因此并不将其进行必要的语法整理。

**tracemonkey 语法树整理后代码：**

```
function foo(){  
    var A;  
    if ( 1===1 ) {  
        A=function (){alert('A')};  
    }else{  
        A=function (){alert('B')};  
    }  
    A();  
}
```

同名函数表达式存在于不同分支语句中，并不产生覆盖/重写效果，实际结果为A。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍





# ENCODING DECLARATION IN THE BROWSER AFFECTS THE FAULT-TOLERANT



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# 编码声明在浏览器内的容错影响

确定一个文档的字符编码的优先级顺序（由高至低）：

1. HTTP "Content-Type" 字段中的 "charset" 参数。  
`Content-Type: text/html; charset=ISO-8859-1`
2. META 声明中 "http-equiv" 为 "Content-Type" 对应的值中的 "charset" 的值。  
`<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">`
3. 元素的 charset 属性。  
`<script charset="iso-8859-1" src="XXX"></script>`

我们通常情况下为页面设定的字符编码信息所指对应到浏览器内部大多是字符编码别名，如 ISO-8859-1。

指定了浏览器无法识别的字符编码别名时，浏览器参照上面的优先级顺序采用更低优先级的字符编码，以此类推。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# 编码声明在浏览器内的容错影响

各浏览器对于没有任何字符编码设定的页面所采用的编码类型：

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<script>
    document.write((document.charset || document.characterSet).toUpperCase());
</script>
</body>
</html>
```

操作系统及浏览器语言均为简体中文。页面中没有设定任何的字符编码信息，页面自身的编码为 GB2312。

IE6 IE7 IE8 Firefox	Chrome Safari	Opera
字符编码 --- GB2312	×Ö·û±àÂë --- ISO-8859-1	字符编码 --- GBK

当页面没有设置任何字符编码信息或者浏览器无法识别 HTTP 头字段以及 META 元素中所声明的字符编码信息时，所有浏览器均会以各自在当前语言版本下的默认字符编码显示页面。

因为页面自身的编码为 GB2312，则 Windows 平台下 IE Firefox Opera 简体中文版的默认字符编码刚好为 GB2312，所以页面中的字符显示正常。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍

# 编码声明在浏览器内的容错影响

看一组特殊的例子：

```
<?php header("Content-Type: text/html; charset=maccyrillic"); ?>
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=b.i.g+5"/>
</head>
<body style="font:24px Tahoma;">
字符編碼 ---
<script>
document.write((document.charset || document.characterSet).toUpperCase());
</script>
</body>
</html>
```

动态页面自身的编码为 **BIG5**。

IE6 IE7 IE8 Firefox	Chrome Safari	Opera
才網綫 --- GB2312	тr≤≈ьsьX --- X-MAC-CYRILLIC	字符編碼 --- BIG5

IE Firefox 无法识别 **maccyrillic**，也无法识别 **b.i.g+5**，继而采用了当前语言版本的默认编码 GB2312，与页面自身的字符编码 BIG5 不相符，导致页面中的文字显示异常。

Chrome Safari 识别 **maccyrillic** 为合法的 **X-MAC-CYRILLIC**。不再理会更低优先级的编码设置。

Opera 无法识别 **maccyrillic**，将 **b.i.g+5** 这种字符编码别名识别为 **BIG5**。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 **WebAdvocate** 很彪悍



# 编码声明在浏览器内的容错影响

出现上述现象的原因主要有三点：

1. 各浏览器的字符编码别名表不尽相同，对同一种编码下的各种别名支持的宽泛程度不一样。像 maccyrillic 这种别名在 Chrome Safari 可以识别为通用的 X-MAC-CYRILLIC<sup>1</sup>，但其他浏览器则会将其视作错误的字符编码信息处理。
2. 各浏览器在匹配页面的字符编码与别名表中的字符编码时，匹配的方式不同。Chrome Safari Opera 会将编码类型的字符串做一个转换，过滤了除英文大小写字符、数字字符之外的字符（isASCIIAlphanumeric）。如 ISO8859\_5 会以转换后的 ISO88595 与别名表中的 ISO-8859-5 转换后的 ISO88595 做比较，b.i.g+5 也会转换为 big5 与别名表中的做比较，所以浏览器可以正确识别这些设置的字符编码为浏览器内部的别名。
3. 各浏览器的默认字符编码不同。

**注 1：**各浏览器均可以识别 X-MAC-CYRILLIC 这种通用的字符编码别名。



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍





# 标准相关

【分享】说说标准——关于样式的特殊性

【分享】说说标准——层叠顺序 (Cascading order) , 看看到底哪个样式在起作用

【分享】说说标准——你真的了解盒子模型(box model)吗?

【分享】说说标准——揭开外边距折叠(Collapsing margins)的面纱

【分享】说说标准——CSS中非常重要的可视化格式模型(visual formatting model)简介



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍



# 兼容性相关

【分享】深入挖掘document.getElementsByTagName()方法的返回值

【分享】CSS Hack的基本原理、常用CSS hack及使用原则

【分享】Firefox中table元素的绝对定位子元素包含块判定错误的bug

【分享】深入挖掘 offsetParent 元素的判定

【分享】零高度的浮动元素是否不影响其他元素定位？



欢迎到CSDN的跨浏览开发论坛发帖  
^\_^ 版主 [WebAdvocate](#) 很彪悍