第 18 章 浏览器检测

学习要点：

1.navigator 对象

2.客户端检测

主讲教师：李炎恢
合作网站：http://www.ibeifeng.com
讲师博客：http://hi.baidu.com/李炎恢

由于每个浏览器都具有自己独到的扩展，所以在开发阶段来判断浏览器是一个非常重要的步骤。虽然浏览器开发商在公共接口方面投入了很多精力，努力的去支持最常用的公共功能；但在现实中，浏览器之间的差异，以及不同浏览器的“怪癖”却是非常多的，因此客户端检测除了是一种补救措施，更是一种行之有效的开发策略。

一. navigator 对象

navigator 对象最早由 Netscape Navigator2.0 引入的 navigator 对象，现在已经成为识别客户端浏览器的事实标准。与之前的 BOM 对象一样，每个浏览器中 的 navigator 对象也都有一套自己的属性。

navigator 对象的属性或方法

属性或方法	说明	IE	Firefox	Safari/Chrome	Opera
appCodeName	浏览器的名称。通常是 Mozilla，即使在非 Mozilla 浏览器中也是如此	3.0+	1.0+	1.0+	7.0+
appName	完整的浏览器名称	3.0+	1.0+	1.0+	7.0+
appMinorVersion	次版本信息	4.0+	-	-	9.5+
appVersion	浏览器的版本。一般不与实际的浏览器版本对应	3.0+	1.0+	1.0+	7.0+
buildID	浏览器编译版本	-	2.0+	-	-
cookieEnabled	表示 cookie 是否启用	4.0+	1.0+	1.0+	7.0+
cpuClass	客户端计算机中使用的 CPU 类型(x86、68K、Alpha、PPC、other)	4.0+	-	-	-
javaEnabled()	表示当前浏览器中是否启用了 Java	4.0+	1.0+	1.0+	7.0+
language	浏览器的主语言	-	1.0+	1.0+	7.0+
mimeTypes	在浏览器中注册的 MIME 类型数组	4.0+	1.0+	1.0+	7.0+
onLine	表示浏览器是否连接到了因特网	4.0+	1.0+	-	9.5+
opsProfile	似乎早就不用了。无法查询	4.0+	-	-	-

oscpu	客户端计算机的操作系统或使用的 CPU	–	1.0+	–	–
platform	浏览器所在的系统平台	4.0+	1.0+	1.0+	7.0+
plugins	浏览器中安装的插件信息的数组	4.0+	1.0+	1.0+	7.0+
preference()	设置用户的首选项	–	1.5+	–	–
product	产品名称（如 Gecko）	–	1.0+	1.0+	–
productSub	关于产品的次要信息（如 Gecko 的版本）	–	1.0+	1.0+	–
registerContentHandler()	针对特定的 MIME 类型讲一个站点注册为处理程序	–	2.0+	–	–
registerProtocolHandler()	针对特定的协议将一个站点注册为处理程序	–	2.0	–	–
securityPolicy	已经废弃。安全策略的名称	–	1.0+	–	–
systemLanguage	操作系统的语言	4.0+	–	–	–
taintEnabled()	已经废弃。表示是否运行变量被修改	4.0+	1.0+	–	7.0+
userAgent	浏览器的用户代理字符串	3.0+	1.0+	1.0+	7.0+
userLanguage	操作系统的默认语言	4.0+	–	–	7.0+
userProfile	借以访问用户个人信息的对象	4.0+	–	–	–
vendor	浏览器的品牌	–	1.0+	1.0+	–
verdorSub	有关供应商的次要信息	–	1.0+	1.0+	–

1.浏览器及版本号

不同的浏览器支持的功能、属性和方法各有不同。比如 IE 和 Firefox 显示的页面可能就会有所略微不同。

```
alert('浏览器名称：' + navigator.appName);
alert('浏览器版本：' + navigator.appVersion);
alert('浏览器用户代理字符串：' + navigator.userAgent);
alert('浏览器所在的系统：' + navigator.platform);
```

2.浏览器嗅探器

浏览器嗅探器是一段程序，有了它，浏览器检测就变得简单了。我们这里提供了一个 browserdetect.js 文件，用于判断浏览器的名称、版本号及操作系统。

调用方式	说明
BrowserDetect.browser	浏览器的名称，例如 Firefox，IE
BrowserDetect.version	浏览器的版本，比如，7、11
BrowserDetect.OS	浏览器所宿主的操作系统，比如 Windows、Linux

```
alert(BrowserDetect.browser);          //名称
alert(BrowserDetect.version);          //版本
alert(BrowserDetect.OS)                //系统
```

3.检测插件

插件是一类特殊的程序。他可以扩展浏览器的功能，通过下载安装完成。比如，在线音乐、视频动画等等插件。

navigator 对象的 plugins 属性，这个一个数组。存储在浏览器已安装插件的完整列表。

属性	含义
name	插件名
filename	插件的磁盘文件名
length	plugins 数组的元素个数
description	插件的描述信息

```
//列出所有的插件名
for (var i = 0; i < navigator.plugins.length; i ++) {
    document.write(navigator.plugins[i].name + '<br />');
}

//检测非 IE 浏览器插件是否存在
function hasPlugin(name) {
    var name = name.toLowerCase();
    for (var i = 0; i < navigator.plugins.length; i ++) {
        if (navigator.plugins[i].name.toLowerCase().indexOf(name) > -1) {
            return true;
        }
    }
    return false;
}

alert(hasPlugin('Flash'));             //检测 Flash 是否存在
alert(hasPlugin('java'))               //检测 Java 是否存在
```

4.ActiveX

IE 浏览器没有插件，但提供了 ActiveX 控件。ActiveX 控件一种在 Web 页面中嵌入对象或组件的方法。

由于在 JS 中，我们无法把所有已安装的 ActiveX 控件遍历出来，但我们还是可以去验证是否安装了此控件。

```
//检测 IE 中的控件
```

```
function hasIEPlugin(name) {
    try {
        new ActiveXObject(name);
        return true;
    } catch (e) {
        return false;
    }
}

//检测 Flash
alert(hasIEPlugin('ShockwaveFlash.ShockwaveFlash'));
```

PS：ShockwaveFlash.ShockwaveFlash 是 IE 中代表 FLASH 的标识符，你需要检查哪种控件，必须先获取它的标识符。

```
//跨浏览器检测是否支持 Flash
function hasFlash() {
    var result = hasPlugin('Flash');
    if (!result) {
        result = hasIEPlugin('ShockwaveFlash.ShockwaveFlash');
    }
    return result;
}

//检测 Flash
alert(hasFlash());
```

5.MIME 类型

MIME 是指多用途因特网邮件扩展。它是通过因特网发送邮件消息的标准格式。现在也被用于在因特网中交换各种类型的文件。

PS：mimeType[]数组在 IE 中不产生输出。

mimeType 对象的属性

属性	含义
type	MIME 类型名
description	MIME 类型的描述信息
enabledPlugin	指定 MIME 类型配置好的 plugin 对象引用
suffixes	MIME 类型所有可能的文件扩展名

```
//遍历非 IE 下所有 MIME 类型信息
for (var i = 0; i < navigator.mimeTypes.length; i++) {
    if (navigator.mimeTypes[i].enabledPlugin != null) {
```

```
            document.write('<dl>');
            document.write('<dd>类型名称：' + navigator.mimeTypes[i].type + '</dd>');
            document.write('<dd>类型引用：' + navigator.mimeTypes[i].enabledPlugin.name +
'</dd>');
            document.write('<dd>类型描述：' + navigator.mimeTypes[i].description + '</dd>');
            document.write('<dd>类型后缀：' + navigator.mimeTypes[i].suffixes + '</dd>');
            document.write('</dl>')
        }
    }
```

二．客户端检测

客户端检测一共分为三种，分别为：能力检测、怪癖检测和用户代理检测，通过这三种检测方案，我们可以充分的了解当前浏览器所处系统、所支持的语法、所具有的特殊性能。

1.能力检测

能力检测又称作为特性检测，检测的目标不是识别特定的浏览器，而是识别浏览器的能力。能力检测不必估计特定的浏览器，只需要确定当前的浏览器是否支持特定的能力，就可以给出可行的解决方案。

```
//BOM 章节的一段程序
var width = window.innerWidth;                    //如果是非 IE 浏览器

if (typeof width != 'number') {                   //如果是 IE，就使用 document
    if (document.compatMode == 'CSS1Compat') {
        width = document.documentElement.clientWidth;
    } else {
        width = document.body.clientWidth;        //非标准模式使用 body
    }
}
```

PS：上面其实有两块地方使用了能力检测，第一个就是是否支持 innerWidth 的检测，第二个就是是否是标准模式的检测，这两个都是能力检测。

2.怪癖检测(bug 检测)

与能力检测类似，怪癖检测的目标是识别浏览器的特殊行为。但与能力检测确认浏览器支持什么能力不同，怪癖检测是想要知道浏览器存在什么缺陷(bug)。

bug 一般属于个别浏览器独有，在大多数新版本的浏览器被修复。在后续的开发过程中，如果遇到浏览器 bug 我们再详细探讨。

```
var box = {
    toString : function () {}                     //创建一个 toString()，和原型中重名了
};
for (var o in box) {
    alert(o);                                     //IE 浏览器的一个 bug，不识别了
}
```

3.用户代理检测

用户代理检测通过检测用户代理字符串来确定实际使用的浏览器。在每一次 HTTP 请求过程中，用户代理字符串是作为响应首部发送的，而且该字符串可以通过 JavaScript 的 navigator.userAgent 属性访问。

用户代理代理检测，主要通过 navigator.userAgent 来获取用户代理字符串的，通过这组字符串，我们来获取当前浏览器的版本号、浏览器名称、系统名称。

PS：在服务器端，通过检测用户代理字符串确定用户使用的浏览器是一种比较广为接受的做法。但在客户端，这种测试被当作是一种万不得已的做法，且饱受争议，其优先级排在能力检测或怪癖检测之后。饱受争议的原因，是因为它具有一定的欺骗性。

```
document.write(navigator.userAgent);                //得到用户代理字符串
```

Firefox14.0.1

Mozilla/5.0 (Windows NT 5.1; rv:14.0) Gecko/20100101 Firefox/14.0.1

Firefox3.6.28

Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.2.28) Gecko/20120306 Firefox/3.6.28

Chrome20.0.1132.57 m

Mozilla/5.0 (Windows NT 5.1) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11

Safari5.1.7

Mozilla/5.0 (Windows NT 5.1) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2

IE7.0

Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

IE8.0

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

IE6.0

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

Opera12.0

Opera/9.80 (Windows NT 5.1; U; zh-cn) Presto/2.10.289 Version/12.00

Opera7.54

Opera/7.54 (Windows NT 5.1; U) [en]

Opera8
Opera/8.0 (Window NT 5.1; U; en)

Konqueror (Linux 集成，基于 KHTML 呈现引擎的浏览器)
Mozilla/5.0 (compatible; Konqueror/3.5; SunOS) KHTML/3.5.0 (like Gecko)

只要仔细的阅读这些字符串，我们可以发现，这些字符串包含了浏览器的名称、版本和所宿主的操作系统。

每个浏览器有它自己的呈现引擎：所谓呈现引擎，就是用来排版网页和解释浏览器的引擎。通过代理字符串发现，我们归纳出浏览器对应的引擎：
IE -- Trident，　　IE8 体现出来了，之前的未体现
Firefox -- Gecko，
Opera -- Presto，　旧版本根本无法体现呈现引擎
Chrome -- WebKit　WebKit 是 KHTML 呈现引擎的一个分支，后独立开来
Safari -- WebKit
Konqueror -- KHTML

由上面的情况，我们需要检测呈现引擎可以分为五大类：IE、Gecko、WebKit、KHTML 和 Opera。

```
var client = function () {                          //创建一个对象

    var engine = {                                  //呈现引擎
        ie : false,
        gecko : false,
        webkit : false,
        khtml : false,
        opera : false,

        ver : 0                                     //具体的版本号
    };

    return {
        engine : engine                             //返回呈现引擎对象
    };
}();                                                //自我执行

alert(client.engine.ie);                            //获取 ie
```

以上的代码实现了五大引擎的初始化工作，分别给予 true 的初值，并且设置版本号为 0。
下面我们首先要做的是判断 Opera，因为 Opera 浏览器支持 window.opera 对象，通过这个对象，我们可以很容易获取到 Opera 的信息。

```
for (var p in window.opera) {                       //获取 window.opera 对象信息
    document.write(p + "<br />");
```

```
}

if (window.opera) {                          //判断 opera 浏览器
    engine.ver = window.opera.version();     //获取 opera 呈现引擎版本
    engine.opera = true;                     //设置真
}
```

接下来，我们通过正则表达式来获取 WebKit 引擎和它的版本号。

```
else if (/AppleWebKit\/(\S+)/.test(ua)) {    //正则 WebKit
    engine.ver = RegExp['$1'];               //获取 WebKit 版本号
    engine.webkit = true;
}
```

然后，我们通过正则表达式来获取 KHTML 引擎和它的版本号。由于这款浏览器基于 Linux，我们无法测试。

```
//获取 KHTML 和它的版本号
else if (/KHTML\/(\S+)/.test(ua) || /Konqueror\/([^;]+)/.test(ua)) {
    engine.ver = RegExp['$1'];
    engine.khtml = true;
}
```

下面，我们通过正则表达式来获取 Gecko 引擎和它的版本号。

```
else if (/rv:([^\)]+)\) Gecko\/\d{8}/.test(ua)) {    //获取 Gecko 和它的版本号
    engine.ver = RegExp['$1'];
    engine.gecko = true;
}
```

最后，我们通过正则表达式来获取 IE 的引擎和它的版本号。因为 IE8 之前没有呈现引擎，所以，我们只有通过"MSIE"这个共有的字符串来获取。

```
else if (/MSIE ([^;]+)/.test(ua)) {          //获取 IE 和它的版本号
    engine.ver = RegExp['$1'];
    engine.ie = true;
}
```

上面获取各个浏览器的引擎和引擎的版本号，但大家也发现了，其实有些确实是浏览器的版本号。所以，下面，我们需要进行浏览器名称的获取和浏览器版本号的获取。

根据目前的浏览器市场份额，我们可以给一下浏览器做检测：IE、Firefox、konq、opera、chrome、safari。

```
var browser = {                              //浏览器对象
    ie : false,
    firefox : false,
    konq : false,
    opera : false,
    chrome : false,
```

```
        safari : false,

        ver : 0,                                    //具体版本
        name : ''                                   //具体的浏览器名称
    };
```

对于获取 IE 浏览器的名称和版本，可以直接如下：

```
    else if (/MSIE ([^;]+)/.test(ua)) {
        engine.ver = browser.ver = RegExp['$1'];    //设置版本
        engine.ie = browser.ie = true;              //填充保证为 true
        browser.name = 'Internet Explorer';         //设置名称
    }
```

对于获取 Firefox 浏览器的名称和版本，可以如下：

```
    else if (/rv:([^\)]+)\) Gecko\/\d{8}/.test(ua)) {
        engine.ver = RegExp['$1'];
        engine.gecko = true;
        if (/Firefox\/(\S+)/.test(ua)) {
            browser.ver = RegExp['$1'];             //设置版本
            browser.firefox = true;                 //填充保证为 true
            browser.name = 'Firefox';               //设置名称
        }
    }
```

对于获取 Chrome 和 safari 浏览器的名称和版本，可以如下：

```
    else if (/AppleWebKit\/(\S+)/.test(ua)) {
        engine.ver = RegExp['$1'];
        engine.webkit = parseFloat(engine.ver);
        if (/Chrome\/(\S+)/.test(ua)) {
            browser.ver = RegExp['$1'];
            browser.chrome = true;
            browser.name = 'Chrome';
        } else if (/Version\/(\S+)/.test(ua)) {
            browser.ver = RegExp['$1'];
            browser.chrome = true;
            browser.name = 'Safari';
        }
    }
```

PS：对于 Safari3 之前的低版本，需要做 WebKit 的版本号近似映射。而这里，我们将不去深究，已提供代码。

浏览器的名称和版本号，我们已经准确的获取到，最后，我们想要去获取浏览器所宿主的操作系统。

```
var system = {                          //操作系统
    win : false,                        //windows
    mac : false,                        //Mac
    x11 : false                         //Unix、Linux
};

var p = navigator.platform;             //获取系统
system.win = p.indexOf('Win') == 0;     //判断是否是 windows
system.mac = p.indexOf('Mac') == 0;     //判断是否是 mac
system.x11 = (p == 'X11') || (p.indexOf('Linux') == 0)      //判断是否是 Unix、Linux
```

PS：这里我们也可以通过用户代理字符串获取到 windows 相关的版本，这里我们就不去深究了，提供代码和对应列表。

Windows 版本	IE4+	Gecko	Opera < 7	Opera 7+	WebKit
95	"Windows 95"	"Win95"	"Windows 95"	"Windows 95"	n/a
98	"Windows 98"	"Win98"	"Windows 98"	"Windows 98"	n/a
NT4.0	"Windows NT"	"WinNT4.0"	"Windows NT 4.0"	"Windows NT 4.0"	n/a
2000	"Windows NT 5.0"	"Windows NT5.0"	"Windows 2000"	"Windows NT 5.0"	n/a
ME	"Win 9X 4.90"	"Win 9x 4.90"	"Windows ME"	"Win 9X 4.90"	n/a
XP	"Windows NT 5.1"	"Windows NT 5.1"	"Windows XP"	"Windows NT 5.1"	"Windows NT 5.1"
Vista	"Windows NT 6.0"	"Windows NT 6.0"	n/a	"Windows NT 6.0"	"Windows NT 6.0"
7	"Windows NT 6.1"	"Windows NT 6.1"	n/a	"Windows NT 6.1"	"Windows NT 6.1"

感谢收看本次教程！

本课程是由北风网(ibeifeng.com)

瓢城 Web 俱乐部(yc60.com)联合提供：

本次主讲老师：李炎恢

我的博客：hi.baidu.com/李炎恢/

我的邮件：yc60.com@gmail.com