**[优化网站设计（二十六）：设计“智能”的事件处理程序](http://www.cnblogs.com/chenxizhang/archive/2013/05/19/3086643.html)**

**前言**

网站设计的优化是一个很大的话题,有一些通用的原则,也有针对不同开发平台的一些建议。这方面的研究一直没有停止过，我在不同的场合也分享过这样的话题。

作为通用的原则，雅虎的工程师团队曾经给出过35个最佳实践。这个列表请参考Best Practices for Speeding Up Your Web Site  （ <http://developer.yahoo.com/performance/rules.html>），同时，他们还发布了一个相应的测试工具Yslow  <http://developer.yahoo.com/yslow/>

我强烈推荐所有的网站开发人员都应该学习这些最佳实践，并结合自己的实际项目情况进行应用。 接下来的一段时间，我将结合ASP.NET这个开发平台，针对这些原则，通过一个系列文章的形式，做些讲解和演绎，以帮助大家更好地理解这些原则，并且更好地使用他们。

**准备工作**

为了跟随我进行后续的学习，你需要准备如下的开发环境和工具

1. Google Chrome 或者firefox ，并且安装 [Yslow](http://yslow.org/ruleset-matrix/) 这个扩展组件.请注意，这个组件是雅虎提供的，但目前没有针对IE的版本。
   1. <https://chrome.google.com/webstore/detail/yslow/ninejjcohidippngpapiilnmkgllmakh>
   2. <https://addons.mozilla.org/en-US/firefox/addon/yslow/>
   3. 你应该对这些浏览器的开发人员工具有所了解，你可以通过按下F12键调出这个工具。
2. Visaul Studio 2010 SP1 或更高版本，推荐使用Visual Studio 2012
   1. <http://www.microsoft.com/visualstudio/eng/downloads>
3. 你需要对ASP.NET的开发基本流程和核心技术有相当的了解，本系列文章很难对基础知识做普及。

**本文要讨论的话题**

这一篇我和大家讨论的是第二十六条原则：[Develop Smart Event Handlers](http://developer.yahoo.com/performance/rules.html#events) （设计“智能”的事件处理程序）。

我故意给“智能”两个字打上了双引号，意思是说，其实这也算不上智能，我们需要了解DOM元素的事件工作机制，就能正常地写出更好的事件处理程序。

对于DOM的事件机制，你可能自认为相当了解了，例如你能熟悉地报出不少事件的名称（例如load,unload,click,change,focus,blur等），记住他们当然是很好的，但这可不是全部。有兴趣的朋友可以看看W3C的标准文档：<http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-flow>。这个标准文档中提到了事件流的概念（Event-flow），并且提到一种所谓的“冒泡（Bubbing）”机制。

那么，什么是“冒泡”机制呢？我们可以通过一个简单的实例来讲解。

下面是一个简单的页面定义，里面有很多个按钮，我们希望用户点击每个按钮的时候，都能弹出一个对话框，并显示当前按钮的文本。

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication4.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div id="testdiv">

<input type="button" value="button 1" />

<input type="button" value="button 2" />

<input type="button" value="button 3" />

<input type="button" value="button 4" />

<input type="button" value="button 5" />

<input type="button" value="button 6" />

<input type="button" value="button 7" />

</div>

</form>

<script src="Scripts/jquery-2.0.0.min.js"></script>

<script type="text/javascript">

$(function () {

$("input[type=button]").click(function (event) {

alert("Button Clicked : " + $(this).val());

});

});

</script>

</body>

</html>

这样做当然是没有问题的，这是最直接和“正常”的用法。但如果细想一下的话，上述的代码，其实是为每个按钮控件都绑定了一个处理程序。如果按钮有很多（例如100个），那么就需要产生100个事件的绑定。过多的事件绑定会对性能有所影响。

利用DOM事件的冒泡机制，我们可以将代码改写成下面这样：

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication4.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div id="testdiv">

<input type="button" value="button 1" />

<input type="button" value="button 2" />

<input type="button" value="button 3" />

<input type="button" value="button 4" />

<input type="button" value="button 5" />

<input type="button" value="button 6" />

<input type="button" value="button 7" />

</div>

</form>

<script src="Scripts/jquery-2.0.0.min.js"></script>

<script type="text/javascript">

$(function () {

**$("#testdiv").click(function (event) {**

**var bt = $(event.target);**

**alert("Div Clicked : " + bt.val());**

**});**

});

</script>

</body>

</html>

我们看到，在这个改进的版本中，没有直接对按钮进行事件绑定，而是为它们的容器控件（DIV)做了一个事件绑定。如果你运行起来，实际上的效果和之前那一次是一样的：点击每个按钮，分别会弹出一个对话框，显示当前按钮的文本。

这是怎么回事呢？总结起来说，DOM元素的**一些事件（例如click)**会按照下面的方式运作的：

1. 用户点击了按钮，首先会去查找按钮上面有没有直接绑定事件处理程序，如果有的话，先执行这个事件处理程序；
2. 然后会尝试查找按钮的上层元素是否有绑定相应的事件处理程序，如果有，则也会执行。
3. 再往上查找，只要有相应的事件注册，都会被执行，直到最顶层的BODY为止。

这就是“冒泡”的意思。同时，这种事件机制还有一种叫法：[事件代理](http://www.bing.com/search?q=javascript+%E4%BA%8B%E4%BB%B6%E4%BB%A3%E7%90%86&go=&qs=n&form=QBLH&pq=javascript+%E4%BA%8B%E4%BB%B6%E4%BB%A3%E7%90%86&sc=0-11&sp=-1&sk=)。

一个附加的问题是：如果我们既在按钮上面订阅了事件，而且也在DIV上面订阅了事件，那么会不会同时都会被触发了。

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication4.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div id="testdiv">

<input type="button" value="button 1" />

<input type="button" value="button 2" />

<input type="button" value="button 3" />

<input type="button" value="button 4" />

<input type="button" value="button 5" />

<input type="button" value="button 6" />

<input type="button" value="button 7" />

</div>

</form>

<script src="Scripts/jquery-2.0.0.min.js"></script>

<script type="text/javascript">

$(function () {

$("input[type=button]").click(function (event) {

alert("Button Clicked : " + $(this).val());

});

$("#testdiv").click(function (event) {

var bt = $(event.target);

alert("Div Clicked : " + bt.val());

});

});

</script>

</body>

</html>

答案是：他们都会被执行。

那么，如果我想在某些情况下，只触发按钮直接订阅的事件，而不触发DIV订阅的事件（阻止将事件冒泡），行不行呢？当然是可以的，你可以添加下面这样的代码

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication4.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div id="testdiv">

<input type="button" value="button 1" />

<input type="button" value="button 2" />

<input type="button" value="button 3" />

<input type="button" value="button 4" />

<input type="button" value="button 5" />

<input type="button" value="button 6" />

<input type="button" value="button 7" />

</div>

</form>

<script src="Scripts/jquery-2.0.0.min.js"></script>

<script type="text/javascript">

$(function () {

$("input[type=button]").click(function (event) {

alert("Button Clicked : " + $(this).val());

**window.event.cancelBubble = true;**

});

$("#testdiv").click(function (event) {

var bt = $(event.target);

alert("Div Clicked : " + bt.val());

});

});

</script>

</body>

</html>

所以，通过本文，我们了解到事件远非我们看到的那么简单。通过理解“冒泡”或者“事件代理”，我们可以将事件处理得更加合理。

值得一说的是，在桌面开发的WPF和Silverlight中，很多事件也同样采用了“冒泡”这样的策略，有兴趣的朋友可以参考 ： <http://msdn.microsoft.com/en-us/library/ms742806.aspx>