**22高级技巧**

**高级函数**

1. 安全的类型检测

typeof 操作符，有一些无法预知的行为，经常会导致检测数据类型时得到不靠谱的结果。Safari（直至第 4 版）在对**正则表达式**应用 typeof 操作符时会返回"function"，因此很难确定某个值到底是不是函数。

instanceof 操作符在存在多个全局作用域（像一个页面包含多个 frame）的情况下，以下代码要返回 true，value 必须是一个数组，而且还必须与 Array 构造函数在同个全局作用域中。（别忘了，Array 是 window 的属性。）

var isArray = value instanceof Array;

在检测某个对象到底是原生对象还是开发人员自定义的对象的时候，也会有问题。出现这个问题的原因是浏览器开始原生支持 JSON 对象了。

解决上述问题的办法都一样。大家知道，在任何值上调用 Object 原生的 toString()方法，都会返回一个[object NativeConstructorName]格式的字符串。

alert(Object.prototype.toString.call(value)); //"[object Array]"

Object 的 toString()方法不能检测非原生构造函数的构造函数名。因此，开发人员定义的任何构造函数都将返回[object Object]。有些 JavaScript 库会包含与下面类似的代码。

var isNativeJSON = window.JSON && Object.prototype.toString.call(JSON) ==

"[object JSON]";

1. 作用域安全的构造函数

当使用 new 调用时，构造函数内用到的 this 对象会指向新创建的对象实例，。问题出在当没有使用 new操作符来调用该构造函数的情况上。由于该 this 对象是在运行时绑定的，所以**直接调用 Person()，this 会映射到全局对象 window 上**，导致错误对象属性的意外增加。[Demo1](demo1.html)

由于 window 的 name 属性是用于识别链接目标和 frame 的，所以这里对该属性的偶然覆盖可能会导致该页面上出现其他错误。这个问题的解决方法就是创建一个作用域安全的构造函数。

首先确认 this 对象是正确类型的实例。如果不是，那么会创建新的实例并返回。请看以下例子：

function Person(name, age, job){

**if (this instanceof Person){**

this.name = name;

this.age = age;

this.job = job;

**} else {**

**return new Person(name, age, job);**

**}**

}

实现这个模式后，你就锁定了可以调用构造函数的环境。如果你**使用构造函数窃取模式的继承且不使用原型链**，那么这个继承很可能被破坏。[Demo2](demo2.html)

this 对象并非 Polygon 的实例，所以会创建并返回一个新的 Polygon 对象。Rectangle 构造函数中的 this 对象并没有得到增长，同时 Polygon.call()返回的值也没有用到，所以 Rectangle 实例中就不会有 sides 属性。

如果构造函数窃取结合使用原型链或者寄生组合则可以解决这个问题。[Demo2](demo2.html)

1. 懒性载入函数

因为浏览器之间行为的差异，多数 JavaScript 代码包含了大量的 if 语句，将执行引导到正确的代码中。看看下面来自上一章的 createXHR()函数。[Demo3](demo3.html)

如果浏览器支持内置 XHR，那么它就一直支持了，那么这种测试就变得没必要了。有两种实现惰性载入的方式，第一种就是在函数被调用时再处理函数（重新定义函数）。但第一次调用函数时会损失性能[Demo3](file:///E:\Mark\learn-mark\js高设\第二十二章高级技巧\demo3.html)

第二种实现惰性载入的方式是在声明函数时就指定适当的函数。这样，第一次调用函数时就不会损失性能了，而在代码首次加载时会损失一点性能。这个例子中使用的技巧是创建一个匿名、自执行的函数，用以确定应该使用哪一个函数实现。（第一次运行之后return function）[Demo3](file:///E:\Mark\learn-mark\js高设\第二十二章高级技巧\demo3.html)

1. 函数绑定

另一个日益流行的高级技巧叫做函数绑定。函数绑定要创建一个函数，可以在特定的 this 环境中以指定参数调用另一个函数。该技巧常常和回调函数与事件处理程序一起使用，以便在将函数作为变量传递的同时保留代码执行环境。

创建了一个叫做 handler 的对象。handler.handleClick()方法被分配为一个 DOM 按钮的事件处理程序。当按下该按钮时，就调用该函数，显示一个警告框。this 对象最后是指向了 DOM 按钮而非 handler（在 IE8 中，this 指向 window。）

var handler = {

message: "Event handled",

handleClick: function(event){

alert(this.message);

}

};

var btn = document.getElementById("my-btn");

EventUtil.addHandler(btn, "click", handler.handleClick);

使用一个闭包来修正这个问题。

EventUtil.addHandler(btn, "click", function(event){

handler.handleClick(event);

});