第 28 章 错误处理与调试

学习要点：

1.浏览器错误报告
2.错误处理
3.错误事件
4.错误处理策略
5.调试技术
6.调试工具

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

JavaScript 在错误处理调试上一直是它的软肋，如果脚本出错，给出的提示经常也让人摸不着头脑。ECMAScript 第 3 版为了解决这个问题引入了 try...catch 和 throw 语句以及一些错误类型，让开发人员更加适时的处理错误。

一．浏览器错误报告

随着浏览器的不断升级，JavaScript 代码的调试能力也逐渐变强。IE、Firefox、Safari、Chrome 和 Opera 等浏览器，都具备报告 JavaScript 错误的机制。只不过，浏览器一般面向的是普通用户，默认情况下会隐藏此类信息。

IE：在默认情况下，左下角会出现错误报告，双击这个图标，可以看到错误消息对话框。如果开启禁止脚本调试，那么出错的时候，会弹出错误调试框。设置方法为：工具->Internet Options 选项->高级->禁用脚本调试，取消勾选即可。

Firefox：在默认情况下，错误不会通过浏览器给出提示。但在后台的错误控制台可以查看。查看方法为：工具->[Web 开发者]->Web 控制台|错误控制台。除了浏览器自带的，开发人员为 Firefox 提供了一个强大的插件：Firebug。它不但可以提示错误，还可以调试 JavaScript 和 CSS、DOM、网络链接错误等。

Safari：在默认情况下，错误不会通过浏览器给出提示。所以，我们需要开启它。查看方法为：显示菜单栏->编辑->偏好设置->高级->在菜单栏中显示开发->显示 Web 检查器|显示错误控制器。

Opera：在默认情况下，错误会被隐藏起来。打开错误记录的方式为：显示菜单栏->查看->开发者工具->错误控制台。

Chrome：在默认情况下，错误会被隐藏起来。打开错误记录的方法为：工具->JavaScript 控制台。

二．错误处理

良好的错误处理机制可以及时的提醒用户，知道发生了什么事，而不会惊慌失措。为此，作为开发人员，我们必须理解在处理 JavaScript 错误的时候，都有哪些手段和工具可以利用。

try-catch 语句

ECMA262 第 3 版引入了 try-catch 语句，作为 JavaScript 中处理异常的一种标准方式。

```
try {                                          //尝试着执行 try 包含的代码
    window.abcdefg();                          //不存在的方法
} catch (e) {                                  //如果有错误，执行 catch，e 是异常对象
    alert('发生错误啦，错误信息为：' + e);      //直接打印调用 toString()方法
}
```

在 e 对象中，ECMA-262 还规定了两个属性：message 和 name，分别打印出信息和名称。

```
alert('错误名称：' + e.name);
alert('错误名称：' + e.message);
```

PS：Opera9 之前的版本不支持这个属性。并且 IE 提供了和 message 完全相同的 description 属性、还添加了 number 属性提示内部错误数量。Firefox 提供了 fileName(文件名)、lineNumber(错误行号)和 stack(栈跟踪信息)。Safari 添加了 line(行号)、sourceId(内部错误代码)和 sourceURL(内部错误 URL)。所以，要跨浏览器使用，那么最好只使用通用的 message。

finally 子句

finally 语句作为 try-catch 的可选语句，不管是否发生异常处理，都会执行。并且不管 try 或是 catch 里包含 return 语句，也不会阻止 finally 执行。

```
try {
    window.abcdefg();
} catch (e) {
    alert('发生错误啦，错误信息为：' + e.stack);
} finally {                                    //总是会被执行
    alert('我都会执行！');
}
```

PS：finally 的作用一般是为了防止出现异常后，无法往下再执行的备用。也就是说，如果有一些清理操作，那么出现异常后，就执行不到清理操作，那么可以把这些清理操作放到 finally 里即可。

错误类型

执行代码时可能会发生的错误有很多种。每种错误都有对应的错误类型，ECMA-262 定义了 7 种错误类型：

1.Error
2.EvalError
3.RangeError
4.ReferenceError
5.SyntaxError
6.TypeError
7.URIError

其中，Error 是基类型(其他六种类型的父类型)，其他类型继承自它。Error 类型很少见，一般由浏览器抛出的。这个基类型主要用于开发人员抛出自定义错误。

PS：抛出的意思，就是当前错误无法处理，丢给另外一个人，比如丢给一个错误对象。

```
new Array(-5);                                  //抛出 RangeError(范围)
```
错误信息为：RangeError: invalid array length（无效的数组的长度）
PS：RangeError 错误一般在数值超出相应范围时触发

```
var box = a;                                    //抛出 ReferenceError(引用)
```
错误信息为：ReferenceError: a is not defined（a 是没有定义的）
PS：ReferenceError 通常访问不存在的变量产生这种错误

```
a $ b;                                          //抛出 SyntaxError(语法)
```
错误信息为：SyntaxError: missing ; before statement（失踪;语句之前）
PS：SyntaxError 通常是语法错误导致的

```
new 10;                                         //抛出 TypeError(类型 )
```
错误信息为：TypeError: 10 is not a constructor（10 不是一个构造函数）
PS：TypeError 通常是类型不匹配导致的

PS：EvalError 类型表示全局函数 eval()的使用方式与定义的不同时抛出，但实际上并不能产生这个错误，所以实际上碰到的可能性不大。

PS：在使用 encodeURI()和 decodeURI()时，如果 URI 格式不正确时，会导致 URIError 错误。但因为 URI 的兼容性非常强，导致这种错误几乎见不到。

```
alert(encodeURI('李炎恢'));
```

利用不同的错误类型，可以更加恰当的给出错误信息或处理。

```
try {
    new 10;
} catch (e) {
    if (e instanceof TypeError) {              //如果是类型错误，那就执行这里
        alert('发生了类型错误，错误信息为：' + e.message);
    } else {
        alert('发生了未知错误！');
    }
}
```

善用 try-catch

在明明知道某个地方会产生错误，可以通过修改代码来解决的地方，是不适合用 try-catch 的。或者是那种不同浏览器兼容性错误导致错误的也不太适合，因为可以通过判断浏览器或者判断这款浏览器是否存在此属性和方法来解决。

```
try {
    var box = document.getElementbyid('box');  //单词大小写错误，导致类型错误
} catch (e) {                                  //这种情况没必要 try-catch
    alert(e);
}
```

```
try {
    alert(innerWidth);                          //W3C 支持，IE 报错
} catch (e) {
    alert(document.documentElement.clientWidth);   //兼容 IE
}
```

PS：常规错误和这种浏览器兼容错误，我们都不建议使用 try-catch。因为常规错误可以修改代码即可解决，浏览器兼容错误，可以通过普通 if 判断即可。并且 try-catch 比一般语句消耗资源更多，负担更大。所以，在万不得已，无法修改代码，不能通过普通判断的情况下才去使用 try-catch，比如后面的 Ajax 技术。

抛出错误

使用 catch 来处理错误信息，如果处理不了，我们就把它抛出丢掉。抛出错误，其实就是在浏览器显示一个错误信息，只不过，错误信息可以自定义，更加精确和具体。

```
try {
    new 10;
} catch (e) {
    if (e instanceof TypeError) {
        throw new TypeError('实例化的类型导致错误！');   //直接中文解释错误信息
    } else {
        throw new Error('抛出未知错误！');
    }
}
```

PS：IE 浏览器只支持 Error 抛出的错误，其他错误类型不支持。

三．错误事件

error 事件是当某个 DOM 对象产生错误的时候触发。

```
addEvent(window, 'error', function () {
    alert('发生错误啦！')
});

new 10;                                         //写在后面

<img src="123.jpg" onerror="alert('图像加载错误！')" />
```

四．错误处理策略

由于 JavaScript 错误都可能导致网页无法使用，所以何时搞清楚及为什么发生错误至关重要。这样，我们才能对此采取正确的应对方案。

常见的错误类型

因为 JavaScript 是松散弱类型语言，很多错误的产生是在运行期间的。一般来说，需要关注 3 种错误：

1.类型转换错误；2.数据类型错误；3.通信错误，这三种错误一般会在特定的模式下或者没有对值进行充分检查的情况下发生。

类型转换错误

在一些判断比较的时候，比如数组比较，有相等和全等两种：

```
alert(1 == '1');                    //true
alert(1 === '1');                   //false
alert(1 == true);                   //true
alert(1 === true);                  //false
```

PS：由于这个特性，我们建议在这种会类型转换的判断，强烈推荐使用全等，以保证判断的正确性。

```
var box = 10;                       //可以试试 0
if (box) {                          //10 自动转换为布尔值为 true
    alert(box);
}
```

PS：因为 0 会自动转换为 false，其实 0 也是数值，也是有值的，不应该认为是 false，所以我们要判断 box 是不是数值再去打印。

```
var box = 0;
if (typeof box == 'number') {       //判断 box 是 number 类型即可
    alert(box);
}
```

PS：typeof box == 'number'这里也是用的相等，没有用全等呀？原因是 typeof box 本身返回的就是类型的字符串，右边也是字符串，那没必要验证类型，所以相等就够了。

数据类型错误

由于 JavaScript 是弱类型语言，在使用变量和传递参数之前，不会对它们进行比较来确保数据类型的正确。所以，这样开发人员必须需要靠自己去检测。

```
function getQueryString(url) {      //传递了非字符串，导致错误
    var pos = url.indexOf('?');
    return pos;
}
alert(getQueryString(1));
```

PS：为了避免这种错误的出现，我们应该使用类型比较。

```
function getQueryString(url) {
    if (typeof url == 'string') {   //判断了指定类型，就不会出错了
        var pos = url.indexOf('?');
        return pos;
    }
}
alert(getQueryString(1));
```

对于传递参数除了限制数字、字符串之外，我们对数组也要进行限制。

```
function sortArray(arr) {
    if (arr) {                      //只判断布尔值远远不够
        alert(arr.sort());
```

```
        }
    }

    var box = [3,5,1];
    sortArray(box);
```

PS：只用 if (arr)判断布尔值，那么数值、字符串、对象等都会自动转换为 true，而这些类型调用 sort()方法比如会产生错误，这里提一下：空数组会自动转换为 true 而非 false。

```
    function sortArray(arr) {
        if (typeof arr.sort == 'function') {          //判断传递过来 arr 是否有 sort 方法
            alert(arr.sort());                        //就算这个绕过去了
            alert(arr.reverse());                     //这个就又绕不过去了
        }
    }

    var box = {                                       //创建一个自定义对象，添加 sort 方法
        sort : function () {}
    };
    sortArray(box);
```

PS：这断代码本意是判断 arr 是否有 sort 方法，因为只有数组有 sort 方法，从而判断 arr 是数组。但忘记了，自定义对象添加了 sort 方法就可以绕过这个判断，且 arr 还不是数组。

```
    function sortArray(arr) {
        if (arr instanceof Array) {                   //使用 instanceof 判断是 Array 最为合适
            alert(arr.sort());
        }
    }

    var box = [3,5,1];
    sortArray(box);
```

通信错误

在使用 url 进行参数传递时，经常会传递一些中文名的参数或 URL 地址，在后台处理时会发生转换乱码或错误，因为不同的浏览器对传递的参数解释是不同的，所以有必要使用编码进行统一传递。

比如：?user=李炎恢&age=100

```
    var url = '?user=' + encodeURIComponent('李炎恢') + '&age=100';   //编码
```

PS：在 AJAX 章节中我们会继续探讨通信错误和编码问题。

五．调试技术

在 JavaScript 初期，浏览器并没有针对 JavaScript 提供调试工具，所以开发人员就想出了一套自己的调试方法，比如 alert()。这个方法可以打印你怀疑的是否得到相应的值，或者放在程序的某处来看看是否能执行，得知之前的代码无误。

```
var num1 = 1;
var num2 = b;                    //在这段前后加上 alert(")调试错误
var result = num1 + num2;
alert(result);
```

PS：使用 alert(")来调试错误比较麻烦，重要裁剪和粘贴 alert(")，如果遗忘掉没有删掉用于调试的 alert(")将特别头疼。所以，我们现在需要更好的调试方法。

将消息记录到控制台

IE8、Firefox、Opera、Chrome 和 Safari 都有 JavaScript 控制台，可以用来查看 JavaScript 错误。对于 Firefox，需要安装 Firebug，其他浏览器直接使用 console 对象写入消息即可。

console 对象的方法

方法名	说明
error(message)	将错误消息记录到控制台
info(message)	将信息性消息记录到控制台
log(message)	将一般消息记录到控制台
warn(message)	将警告消息记录到控制台

```
console.error('错误！');          //红色带叉
console.info('信息！');           //白色带信息号
console.log('日志！');            //白色
console.warn('警告！');           //黄色带感叹号
```

PS：这里以 Firefox 为标准，其他浏览器会稍有差异。

```
var num1 = 1;
console.log(typeof num1);        //得到 num1 的类型
var num2 = 'b';
console.log(typeof num2);        //得到 num2 的类型
var result = num1 + num2;
alert(result);                   //结果是 1b，匪夷所思
```

PS：我们误把 num2 赋值成字符串了，其实应该是数值，导致最后的结果是 1b。那么传统调试就必须使用 alert(typeo num1)来看看是不是数值类型，比较麻烦，因为 alert()会阻断后面的执行，看过之后还要删，删完估计一会儿又忘了，然后又要 alert(typeof num1)来加深印象。如果用了 console.log 的话，所有要调试的变量一目了然，也不需要删除，放着也没事。

将错误抛出

之前已经将结果错误的抛出，这里不在赘述。

```
if (typeof num2 != 'number') throw new Error('变量必须是数值！');
```

六．调试工具

IE8、Firefox、Chrome、Opera、Safari 都自带了自己的调试工具，而开发人员只习惯了 Firefox 一种，所以很多情况下，在 Firefox 开发调试，然后去其他浏览器做兼容。其实 Firebug 工具提供了一种 Web 版的调试工具：Firebug lite。

以下是网页版直接调用调试工具的代码：直接复制到浏览器网址即可。

javascript:(function(F,i,r,e,b,u,g,L,I,T,E){if(F.getElementById(b))return;E=F[i+'NS']&&F.documentElement.namespaceURI;E=E?F[i+'NS'](E,'script'):F[i]('script');E[r]('id',b);E[r]('src',I+g+T);E[r](b,u);(F[e]('head')[0]||F[e]('body')[0]).appendChild(E);E=new%20Image;E[r]('src',I+L);})(document,'createElement','setAttribute','getElementsByTagName','FirebugLite','4','firebug-lite.js','releases/lite/latest/skin/xp/sprite.png','https://getfirebug.com/','#startOpened');

还有一种离线版，把 firebug-lite 下载好，载入工具即可，导致最终工具无法运行，其他浏览器运行完好。虽然 Web 版本的 Firebug Lite 可以跨浏览器使用 Firebug，但除了 Firefox 原生的之外，都不支持断点、单步调试、监视、控制台等功能。好在，其他浏览器自己的调试器都有。

PS：Chrome 浏览器必须在服务器端方可有效。测试也发现，只能简单调试，如果遇到错误，系统不能自动抛出错误给 firebug-lite。

1.设置断点

我们可以选择 Script(脚本)，点击要设置断点的 JS 脚本处，即可设置断点。当我们需要调试的时候，从断点初开始模拟运行，发现代码执行的流程和变化。

2.单步调试

设置完断点后，可以点击单步调试，一步步看代码执行的步骤和流程。上面有五个按钮：

重新运行：重新单步调试

断继：正常执行代码

单步进入：一步一步执行流程

单步跳过：跳到下一个函数块

单步退出：跳出执行到内部的函数

3.监控

单击“监控”选项卡上，可以查看在单步进入是，所有变量值的变化。你也可以新建监控表达式来重点查看自己所关心的变量。

4.控制台

显示各种信息。之前已了解过。

PS：其他浏览器除 IE8 以上均可实现以上的调试功能，大家可以自己常识下。而我们主要采用 Firebug 进行调试然后兼容到其他浏览器的做法以提高开发效率。

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