# 错误处理与调试

ECMAScript第三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，让开发人员能更加适时地处理错误。

## 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，各大浏览器都具备报告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，需要在控制台查看。

## 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定义了七种错误类型：①Error； ②EvalError； ③RangeError； ④ReferenceError； ⑤SyntaxError； ⑥TypeError； ⑦URIError。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 0；//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('高寒') ) ；

利用不同的错误类型，可以更加恰当地给出错误信息或处理。

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

## 错误处理

ECMA第三版引入了try-catch语句，作为JavaScript中处理异常的一种标准方式。

当 JavaScript 引擎执行 JavaScript 代码时，有可能会发生各种错误。当错误发生时，JavaScript 引擎通常会停止（影响其后面代码的执行），并生成一个错误消息。

try {

尝试执行这个块中的代码，没有错，不会执行 catch{}；  
 如果有错误，则会执行 catch{},并且传入错误信息参数

} catch (error) {

}

<**script**>  
  
 **try** { *// 尝试执行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* ***console***.log(**`发生错误了，错误信息为：**${e}**`**); *// 直接打印调用toString()方法* }  
  
</**script**>

在e对象中，ECMA还规定了两个属性：message 和 name， 分别打印出信息和名称。  
console.log(`发生错误了，错误名称为：${e.name}`);   
console.log(`发生错误了，错误信息为：${e.message}`);

还有一个栈跟踪信息：stack  
console.log(`栈跟踪信息：${e.stack}`);

<**script**>  
 *alert*(a);  
 *alert*(**'hello'**);  
</**script**>

结果：因为 a is not defined ,报错了，停止运行后面的代码

<**script**>  
 **try**{  
 *alert*(1);  
 }**catch**(error){  
 *alert*(**'好像有错'**);  
 }  
 *alert*(**'hello'**);  
</**script**>

结果：弹出 1 ，弹出 hello  
try里面没有错，不会执行 catch里面的代码

<**script**>  
 **try**{  
 *alert*(a);  
 }**catch**(error){  
 *alert*(**'好像有错，错误信息：'** + error);  
 }  
 *alert*(**'hello'**);  
</**script**>

结果：弹出 好像有错 ，弹出 hello  
try里面有错，会执行 catch里面的代码，且不影响后面的代码执行。

### 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。

<**script**>  
 **try** {  
 ***window***.abcdefg();  
 } **catch** (e) {  
 ***console***.log(**`发生错误啦，错误信息为：**${e.**stack**}**`**);  
 } **finally** { *// 总是会被执行* ***console***.log(**'我都会执行！'**);  
 }  
</**script**>

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。

### 抛出错误

throw 语句允许我们创建自定义错误。

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。

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

PS：IE浏览器只支持Error抛出的错误，其他错误类型不支持

## 错误事件

error事件是当某个DOM对象产生错误的时候触发。

<**img src="123.jpg" alt="" onerror="**alert(**'图像加载错误'**)**"**>  
<**script**>  
 window.addEventListener(**'error'**,**function**(){  
 console.log(**'发生错误啦！'**);  
 });  
 **new** 10;  
</**script**>

## 调试工具

IE 8、Fie fox、Chrome、Opera， Safari都自带了自己的调试工具，

### 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。

### 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：

1. 重新运行：重新单步调试；  
   (2) 断继：正常执行代码；  
   (3) 单步进入：一步一步执行流程；  
   (4) 单步跳过：跳到下一个函数块；  
   (5) 单步退出：跳出执行到内部的函数。

### 监控

单击“监控”选项卡上，可以查看在单步进入时，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。

### 控制台

显示各种信息。之前已了解过。

PS：其他浏览器除IE8以上均可实现上述的调试功能，大家可以自己尝试一下。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。

示例

<script type="text/javascript">  
 var *sum* = 0;  
 for (let *i* = 0; *i* < 5; *i*++) {  
 *sum* += *i*;  
 }  
 alert(*sum*);  
</script>