# [JavaScript try-catch语句（错误处理）](https://www.cnblogs.com/iceflorence/p/6593139.html)

错误处理在处理程序设计中的重要性是毋庸置疑的，任何有影响力的web应用程序都需要一套完善的错误处理机制。当然，大多数佼佼者确实做到了这一点，但通常只有服务器端应用程序才能做到如此。实际上，服务器端团队往往会在错误处理机制上投入较大精力，通常要考虑按照类型、频率，或者其他重要的标准对错误进行分类。这样一来，开发人员就能够理解用户在使用简单数据库查询或者报告生成脚本时，应用程序可能会出现的问题。

虽然客户端应用程序的错误处理也同样重要，但真正受到重视，还是最近几年的事。实际上，我们要面对这样一个不争的事实：使用web的绝大多数人都不是技术高手，其中甚至很多人根本就不明白浏览器到底是什么，更不用说让他们说喜欢哪一个了。每个浏览器在发生JavaScript错误时的行为都或多或少有一些差异。有的会显示小图标，有的则什么动静都没有，浏览器对JavaScript错误的这些默认行为对最终用户而言，毫无规律可循。最理想的情况下，用户遇到错误搞不清为什么，他们会再试着重做一次，最糟糕的情况下，用户会恼羞成怒，一去不复返了。良好的错误处理机制可以让用户及时得到提醒，知道到底发生了什么事，因而不会惊慌失措。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

ECMA-262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。基本的语法如下所示，显而易见，这与Java中的try-catch语句是完全相同的：

try {

//可能会导致错误的代码

} catch (error) {

//在错误发生时怎么处理

}

 也就是说，我们应该把所有可能会抛出错误的代码都放在try语句快中，而把那些用于错误处理的代码放在catch块中，例如：

try {

window.someNonexistentFunction(); //调用不存在的函数

} catch (error) {

alert('An error happened!');

}

 如果try块中的任何代码发生了错误，就会立即退出代码执行过程，然后接着执行catch块，此时，catch块会接收到一个包含错误信息的对象。与在其他语言中不同的是，即使你不想使用这个错误对象，也要给它起个名字。这个对象中包含的实际信息会因浏览器而异，但共同的是有一个保存着错误信息的message属性。ECMA-262还规定了一个保存着错误类型的name属性，当前所有浏览器都支持这个属性（Opera9之前的版本不支持这个属性）。因此，在发生错误时，就可以像下面这样实事求是地显示浏览器给出的信息：

try {

window.someNonexistentFunction(); //调用不存在的函数

} catch (error) {

alert(error.message);

}

 这个例子在向用户显示错误信息时，使用了错误对象的message属性，这个message属性是唯一一个能保证所有浏览器都支持的属性，除此之外，IE、Firefox、Safari、Chrome以及Opera都为事件对象添加了其他相关信息。IE添加了与message属性完全相同的description属性，还添加了保存着内部错误数量的number属性。Firefox添加了fileName、lineNumber和stack（包含栈跟踪信息）属性。Safari添加了line（表示行号）、sourceId（表示内部错误代码）和sourceUrl属性。当然，在跨浏览器编程时，最好还是只使用message属性。

## **finally子句**

虽然在try-catch语句中是可选的，但finally子句一经使用，其代码无论如何都会执行。换句话说，try语句块中的代码全部正常执行，finally子句会执行；如果因为出错而执行了catch语句块，finally子句照样还会执行。只要代码中包含finally子句，则无论try或catch语句块中包含什么样的代码——甚至return语句，都不会阻止finally子句的执行。来看下面这个函数：

function testFinally() {

try {

return 2;

} catch (error) {

return 1;

} finally {

return 0;

}

}

testFinally(); //0

 这个函数在try-catch语句的每一部分都放了一条return语句。表面上看，调用这个函数会返回2，因为返回2个return语句位于try语句块中，而执行该语句又不会出错。可是，由于最后还有一个finally子句，结果就会导致该return语句被忽略，也就是说，调用这个函数只能返回0，如果把finally子句拿掉，这个函数将返回2.（请读者务必要记住，只要代码中包含finally子句，那么无论try还是catch语句块中的return语句都将被忽略。因此，在使用finally子句之前，一定要非常清楚你想要代码怎么样）

## 错误类型

执行代码期间可能会发生的错误有多种类型，每种错误都有对应的错误类型，而当错误发生时，就会抛出相应类型的错误对象。ECMA-262定义了下列7种错误类型：

01, Error

02, EvalError

03, RangeError

04, ReferenceError

05, SyntaxError

06, TypeError

07, URIError

其中，Error是基类型，其他错误类型都继承自该类型，因此，所有错误类型共享了一组相同的属性（错误对象中的方法全是默认的对象方法）。Error类型的错误很少见，如果有也是浏览器抛出的；这个基类型的主要目的是供开发人员抛出自定义错误。

EvalError类型的错误是在使用eval()函数而发生异常时抛出。ECMA-262中对这个错误有如下描述：“表示全局函数eval的使用方式与其定义不相符。“除此之外，并没有救到底什么情况会引发这种错误给出说明。在实际开发中碰到这种错误的可能性并不大。

RangeError类型的错误会在数值超出相应范围时触发。例如，在定义数组时，如果指定了数组不支持的项数（如-20或Number.MAX\_VALUE），就会触发这种错误。下面是具体的例子：

var items1 = new Array(-20);

//VM77:1 Uncaught RangeError: Invalid array length(…)

var items2 = new Array(Number.MAX\_VALUE);

//VM79:1 Uncaught RangeError: Invalid array length(…)

JavaScript中经常会出现这种范围错误。

在找不到对象的情况下，会发生ReferenceError（这种情况下，会直接导致人所共知的“object expected"浏览器错误）。通常，在访问不存在的变量时，就会发生这种错误，例如：

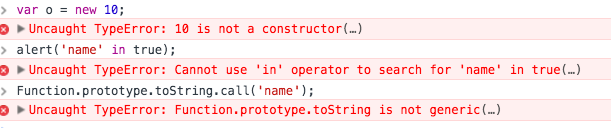
var obj = x;

//VM112:1 Uncaught ReferenceError: x is not defined(…) 在x未声明的情况下抛出ReferenceError

至于SyntaxError，当我们把语法错误的JavaScript字符串传入eval()函数时，就会导致此类错误，例如：

eval('a ++ b'); //VM114:1 Uncaught SyntaxError: Unexpected identifier(…)

TypeError类型在JavaScript中经常用到，在变量中保存着意外的类型时，或者在访问不存在的方法时，都会导致这种错误。错误的原因虽然多种多样，但归根结底还是由于在执行特定于类型的操作时，变量的类型并不符合要求所致。下面来看几个例子：



最常发生类型错误的情况，就是传递给函数的参数事先未经检查，结果传入类型与预期类型不相符。

在使用encodeURI()或decodeURI()，而URI格式不正确时，就会导致URIError错误，这种错误也很少见，因为前面说的这两个函数的容错性非常高。

利用不同的错误类型，可以熟悉更多有关异常的信息，从而有助于对错误作出恰当的处理，要想知道错误的类型，可以像下面这样在try-catch语句的catch语句中使用instanceof操作符：

try {

someFunction();

} catch (error) {

if (error instanceof TypeError) {

//处理类型错误

} else if (error instanceof ReferenceError) {

//处理引用错误

} else {

//处理其他类型的错误 }

}

 在跨浏览器编程中，检查错误类型是确定处理方式的最简便途径，包含在message属性中的错误消息会因浏览器而异。

## 善用try-catch

当try-catch语句中发生错误时，浏览器会认为错误已经被处理了。因而不会通过前面讨论的机制记录或报告错误。对于那些不要求用户懂技术，也不需要用户理解错误的Web应用程序，这应该说是个理想的结果。不过，try-catch能够让我们实现自己的错误处理机制。

使用try-catch最适合处理那些我们无法控制的错误，假设你在使用一个大型的JavaScript库中的函数，该函数可能会有意无意地抛出一些错误。由于我们不能修改这个库的源代码，所以大可将对该函数的调用放在try-catch语句当中。万一有什么错误发生，也好恰当地处理它们。

在明明白白地知道自己的代码会发生错误时，再使用try-catch语句就不太合适了。例如，如果传递给函数的参数是字符串而非数值，就会造成函数出错，那么就应该先检查函数的类型，然后再决定如果去做。在这种情况下，不应该使用try-catch语句。

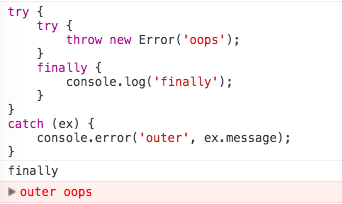
## try-catch语句执行顺序

看下面的例子：

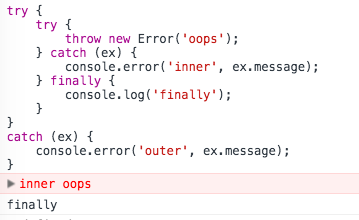


执行顺序为：首先执行try语句块中的代码，如果抛出异常，接着执行catch语句块中代码，如果没有异常，catch语句块中代码将会被忽略，但不管是否有异常，最后最会执行finally子句。try后面必须接着一个catch或者finally，也就是说JavaScript中的try-catch可以有3中组合形式。即try-catch、try-finally、try-catch-finally三种形式。

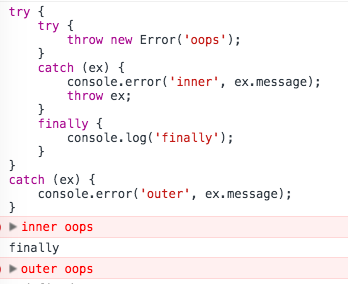
try-catch一般的应用场景大家都比较熟悉，下面来看几个嵌套的例子：



上面这个例子中，最外部的try语句块中嵌套了一个try-finally语句，内部的try语句中抛出了一个异常，但是内部没有catch语句块，所以会执行最近的一个catch语句块，但是在跳出外部try包含语句块之前，需要先执行内部的finally语句块中的代码，所以最后的结果如上图所示。再看一个例子：



这个例子中，内部嵌套的语句块中有catch语句，所以当内部try语句块中抛出异常时，会接着执行内部的catch语句块，然后执行finally子句。由于异常已经在内部处理完成，所以外部的catch语句块会被忽略，所以最终结果如上所示。再看一个例子：



这个例子在上面例子的基础上，内部的catch语句块中又抛出了一个异常，所以，在执行完相应语句后，会接着执行外部的catch语句，结果如上所示。

## 抛出错误

与try-catch语句相配的还有一个throw操作符，用于随时抛出自定义错误。抛出错误时，必须要给throw操作符指定一个值。这个值是什么类型，没有要求。下列代码都是有效的。

throw 12345;

throw 'Hello world!';

throw true;

throw { name: 'JavaScript'};

 在遇到throw操作符时，代码会立即停止执行。仅当有try-catch语句捕获到被抛出的值时，代码才会继续执行。

通过使用某种内置错误类型，可以更真实地模拟浏览器错误。每种错误类型的构造函数接受一个参数，即实际的错误信息。下面是一个例子：

throw new Error('Something bad happened.');

IMG_265

这行代码抛出了一个通用错误，带有一条自定义错误信息。浏览器会像处理自己生成的错误一样，来处理这行代码抛出的错误。换句话说，浏览器会以常规方式报告这一错误，并且会显示这里的自定义错误类型。像下面使用其他错误类型，也可以模拟出类似的浏览器错误：

throw new SyntaxError("I don't like your syntax.");

throw new TypeError("what type of variable do you take me for?");

throw new RangeError("Sorry, you just don't have the range.");

throw new EvalError("That doesn't evaluate.");

throw new URIError("Uri, is that you?");

throw new ReferenceError("You didn't cite your references properly.");

 在创建自定义错误消息时，最常用的错误类型是Error、RangeError、ReferenceError和TypeError。另外，利用原型链还可以通过继承Error来创建自定义错误类型。此时，需要为新创建的错误类型指定name和message属性。来看一个例子：



浏览器对待继承自Error的自定义错误类型，就像对待其他错误类型一样。如果要捕获自己抛出的错误并且把它与浏览器错误区别对待的话，创建自定义错误是很有用的。（IE只有在抛出Error对象的时候才会显示自定义错误信息。对于其他类型，它都无一例外地显示"exception thrown and not caught"（抛出了异常，且未被捕获））。