17.5 异常处理

异常处理语句包括 throw、try、catch、finally，下面详细介绍。

17.5.1 异常概述

异常（Exception）是一个信号，提示代码发生了超出常规预设的行为或结果，异常也可能是程序发生错误的一种征兆。JavaScript 有一套完善的异常处理机制，保证程序不因为异常而崩溃。

异常处理机制就是一套应对 JavaScript 代码发生错误时的处理方法，这套方法被封装在 try/catch/finally 结构中，把代码放在这个结构中执行就可以避免异常发生

JavaScript 把所有可能的错误分门别类进行整理，并把它们封装在不同的对象中，这就是异常的种类。JavaScript 内置的异常对象包括 Error、EvalError、RangeError、SyntaxError、TypeError、ReferenceError 和 URIError。具体说明如表 E17.14 所示。

表 E17.14 JavaScript 内置异常对象

异常类型	说明
Error	普通异常。通常与 throw 语句和 try/catch 语句一起使用。利用属性 name 可以声明或了解异常的类型，利用 message 属性可以设置和读取的异常的详细信息
EvalError	在不正确使用 eval()方法时抛出
SyntaxError	抛出语法错误
RangeError	在数字超出合法范围时抛出
ReferenceError	在读取不存在的变量时抛出
TypeError	当一个值的类型错误时抛出该异常
URIError	由 URl 的编码和解码方法抛出

除了内置异常对象外，JavaScript 允许用户使用自定义异常对象。

17.5.2 throw 语句

throw 语句能够主动抛出一个异常，告诉系统发生了异常状况或错误。throw 语句的语法格式如下：

```
throw expression;
```

expression 可以是任意类型的表达式，一般常用它来声明 Error 对象或者 Error 子类的一个实例。

【示例】在下面循环结构中，定义了一个异常并使用 throw 语句把它抛出来，这样当循环变量大于 5 时，系统会自动弹出一个编译错误，提示“循环变量的值大于 5 了”的错误信息。

```
for(var i = 0;i<10;i++){
    if(i>5)
        throw new Error("循环变量的值大于 5 了");  // 定义并抛出一个异常
}
```

在抛出异常时，JavaScript 解释器会停止程序的正常执行，并跳转到与其最近的异常处理器（catch 结构）。如果解释器没有找到异常处理器，则会检查上一级的 catch 结构，并依此类推，直到找到一个异常处理器为止。如果在程序中没有找到任何异常处理器，将会视其为错误并显示出来。

17.5.3 try/catch/finally 语句

不管是系统抛出，还是用户有意抛出异常，都需要捕获（catch）异常，以便采取适当的动作把程序从异常状态恢复到正常运行状态。

try/catch/finally 语句是 JavaScript 异常处理器，其中 try 从句负责指明需要处理的代码块。catch 从句负责捕获异常，并决定应对之策。finally 从句负责后期处理工作，如清除代码、释放资源等。不管异常是否发生，finally 从句最后都是要执行的。

整个异常处理的结构和从句之间的相互关系如下：

```
try
{
    // 调试代码块
}
catch(e)
{
    // 捕获异常并进行处理
}
finally
{
    // 后期事务处理
}
```

正常情况下，程序按顺序执行 try 从句中的代码，如果没有异常发生，将会忽略 catch 从句，跳转到 finally 从句中继续执行。如果在 try 从句中发生运行时错误或者使用 throw 语句主动抛出异常，则执行 catch 从句代码块，在该从句中通过参数变量引用抛出的 Error 对象或者其他值，同时定义处理异常的方法，或者忽略不计，或者再次抛出异常等。

注意，在异常处理结构中，大括号不是复合语句的一部分，而是异常处理结构的一部分，任何时候都不能够省略这些大括号。

【示例 1】在下面的代码中，先在 try 从句中制造一个语法错误，即字符串没有加引号，然后在 catch 从句中利用参数变量 b 获取 Error 对象的引用，然后提示错误的详细信息，最后在 finally 从句中弹出正确的信息。

```
try{                        // 测试代码
    alert(a);               // 制作语法错误
}
catch(b){                   // 捕获错误
    alert(b.message);       // 提示错误信息
}
finally{                    // 异常后期处理
   alert("a");              // 提示正确值
}
```

【示例 2】在异常处理结构中，catch 和 finally 从句是可选项目，可以根据需要省略，但在正常情况下必须包含 try 和 catch 从句。把上面示例可以精简为：

```
try{
    alert(a);
}
catch(b){}
```

finally 从句比较特殊，不管 try 语句是否完全执行，finally 语句最后都必须要执行，即使使用了跳转语句跳出了异常处理结构，也必须在跳出之前先执行 finally 从句。

如果没有 catch 从句，JavaScript 在执行完 try 从句之后，继续执行 finally 从句，如果发生异常，会继续沿着语法结构链向上查找上一级 catch 从句。

try/catch 语句可以相互嵌套，甚至可以在内层 try/catch 语句中又嵌套另一个内层 try/catch 语句，以及在该内层 try/catch 语句中再嵌套一个 try/catch 语句，嵌套的层数取决于实际代码的意义。

为什么需要使用嵌套的try/catch语句呢？因为使用嵌套的try/catch语句，可以逐步处理内层的try/catch语句抛出的异常。

【示例 3】下面代码就是一个多层嵌套的异常结构，在处理一系列的异常时，内层的 catch 子句通过将异常抛出，就可以将异常抛给外层的 catch 子句来处理。

```
try{                        // 外层异常处理结构
    try{                    // 内层异常处理结构
        alret("Hi");        // 错误引用方法
    }
    catch(exception){
        var e;
```

```
        if ( ! exception.description){        // 兼容非 IE 浏览器
            e = exception.name;               // 获取错误名称
        }
        else{    // 兼容 IE 浏览器
            e = exception.description;        // 获取错误描述信息
        }
        if (e == "Object expected" || e == "ReferenceError"){
          // 如果是此类错误信息，则提示这样信息
            alert("内层 try/catch 能够处理这个错误");
        }
        else{                                 // 否则再一次抛出一个异常
            throw exception;
        }
    }
}
catch(exception){                             // 获取内层异常处理结构中抛出的异常
    alert("内层 try/catch 不能够处理这个错误");
}
```