js实践篇：例外处理Try{}catch(e){}

程序开发中，编程人员经常要面对的是如何编写代码来响应错误事件的发生，即例外处理（exception handlers）。如果例外处理代码设计得周全，那么最终呈现给用户的就将是一个友好的界面。否则，就会让访问者对莫名的现象感到真正的“意外”。

一、什么是例外处理

当JavaScript程序在运行中发生了诸如数组索引越界、类型不匹配或者语法错误时，JavaScript解释器就会引发例外处理。ECMAScript定义了六种类型的错误，除此之外，我们可以使用Error对象和throw语句来创建并引发自定义的例外处理信息。

二、例外处理技术的优点

通过运用例外处理技术，我们可以实现用结构化的方式来响应错误事件的发生，让例外处理代码与正常脚本代码科学分离，最终使我们能够集中精力编写完成主要功能的核心程序。

三、使用 try…catch…finally 执行例外处理

在JavaScript中，我们使用try…catch…finally语句来执行例外处理，即通过它来捕捉错误发生后导致的例外或者执行throw语句产生的例外。它的基本语法如下：

```
try {
    // 此处是可能产生例外的语句
    } catch(error) {
    // 此处是负责例外处理的语句
    } finally {
    // 此处是出口语句
    }
```

上述代码中，try块中的语句首先被执行。如果运行中发生了错误，控制就会转移到位于catch块中语句，其中括号中的error参数被作为例外变量传递。否则，catch块的语句被跳过不执行。无论是发生错误时catch块中的语句执行完毕，或者没有发生错误try块中的语句执行完毕，最后将执行finally块中的语句。

下面我们来看一个例子：

```
<script language="javascript">
try {
        document.writeln("开始执行try块语句 ---> ")
        document.writeln("还没有发生例外 ---> ")
         alert((prompt("输入一个值：  ","")))
 } catch(err) {
         document.writeln("捕捉到例外，开始执行catch块语句 --->");
         document.writeln("错误名称: " + err.name+" ---> ");
         document.writeln("错误信息: " + err.message+" ---> ");
} finally {
         document.writeln("开始执行finally块语句")
}
</script>
```

我们输入abc，然后确定，输出结果如下：

“开始执行try块语句 ---> 还没有发生例外 ---> 捕捉到例外，开始执行catch块语句 ---> 错误名称: TypeError ---> 错误信息: 'abc' 未定义 ---> 开始执行finally块语句”

上述例程以try块语句开始，当输出信息“还没有发生例外”后，弹出输入对话框，要求用户输入一个数值，当我们输入非法的信息"abc"后，就引发了一个例外，所以剩下的try块中的语句将被跳过而开始执行catch块语句。Catch块开始的err参数作为这个例外的错误对象，它具有name和message两

个属性。最后，执行finally块的语句。

我们看到，由于没有错误发生，当try块的语句执行完毕后，catch块语句被跳过，出现一个窗口显示输入的数值，最后执行了finally块的语句。

四、try...catch...finally的变形

try…catch…finally语句有两种变形应用，即try…catch或者try…finally。

try…catch这种结构最常见，它的执行过程是：当没有例外发生执行完毕try块语句后或者发生例外执行完catch块语句后，控制将转移到整个try…catch结构后面的语句。请看下面的例子：

```
try {
        document.writeln("Beginnng the try block")
        document.writeln("No exceptions yet")
// Create a syntax error
("6 + * 3")
document.writeln("Finished the try block with no exceptions")
} catch(err) {
        document.writeln("Exception caught, executing the catch block")
        document.writeln("Error name: " + err.name)
        document.writeln("Error message: " + err.message)
}
document.writeln("Executing after the try-catch statement")
```

如果是try…finally结构，那么当发生例外时，由于没有catch块语句来捕捉错误，所以最终finally块的语句也不会被执行。因此，这种结构在实际应用中很少见。

五、例外的表现形式：Error对象

在JavaScript，例外是作为Error对象出现的。Error对象有两个属性：name属性表示例外的类型，message属性表示例外的含义。根据这些属性的取值，我们可以决定处理例外的方式，比如：

```
function Text() {
try {
        alert((prompt("Enter JavaScript to uate:","")))
} catch(err) {
        if(err.name == "SyntaxError") alert("Invalid expression")
        else alert("Cannot uate")
}
}
```

上面的代码将对用户输入的内容进行表达式求值，然后显示出来。如果在求值过程中发生了SyntaxErroe类型错误，那么就会显示给用户“Invalid expression”的信息；否则，用户得到信息“Cannot uate”。

Error.name的取值一共有六种，如下：

Error：()的使用与定义不一致
RangeError：数值越界
ReferenceError：非法或不能识别的引用数值
SyntaxError：发生语法解析错误
TypeError：操作数类型错误
URIError：URI处理函数使用不当

六、定制例外信息

上述的六种Error类型基本上覆盖了脚本程序运行时所可能发生的错误。除了这些类型以外，我们还可以使用Error构造器来自定义例外类型，其语法如下：

```
myError = new Error(msg)
```

其中msg参数表示所定义的新例外的message属性值。同时，我们还可以创建新的对象类型以作为Error的子类型：

```
function MyError(msg) {
this.name = "MyError"
this.message = msg
}
MyError.prototype = new Error;
```

然后，我们就可以创建自定义错误子类的实例：

```
myError = new MyError("My error message")
```

七、触发例外

创建一个Error对象后，就可以使用throw语句来触发相应的例外。Throw的语法如下：

```
throw errObj
```

errObj必须是一个Error对象或者Error的子类型。在try块代码中触发一个例外后，控制将直接转入catch块。

下面的代码中，在try块中触发了一个例外，设置例外信息为“oops”，然后控制转移到catch块：

```
var s
try {
        s = "one "
```

```
        throw new Error("oops")
        s += "two"
    } catch(err) {
        s += err.message
    }
    s += " three"
    alert(s)
```

编写代码来触发例外的优点很多，比如有利于自定义错误类型，快速转入catch块执行，以及下面要介绍的在嵌套例外中将错误传递到外层。

八、嵌套例外处理

JavaScript支持多层次的嵌套例外处理。一般情况下，我们可以在内部例外处理的catch代码块中捕捉并处理错误，然后再次触发例外，这样就可进一步在外部例外处理的catch代码块中做更加深入的处理。下面来看看一个嵌套例外处理的例子：

```
var inner;
var outer;
try {
        document.writeln("Beginning outer try block, no exceptions yet");
try{
        document.writeln("Beginning inner try block, no exceptions yet");
        // 生成一个引用错误
        document.writeln(undefinedVariable)
        document.writeln("Finished inner try block with no exceptions");
} catch(inner) {
// 内部例外处理
        document.writeln("Exception caught, beginning inner catch block");
        document.writeln("Error type: " + inner.name);
        document.writeln("Error message: " + inner.message);
        throw inner;
        document.writeln("No exceptions thrown in inner catch block");
```

```
} finally {
        document.writeln("Executing inner finally block");
}
        document.writeln("Finished outer try block with no exceptions");
} catch(outer) {
        // 外部例外处理
        document.writeln("Exception caught, beginning outer catch block");
        document.writeln("Error type: " + outer.name);
        document.writeln("Error message: " + outer.message);
} finally {
        document.writeln("Executing outer finally block");
}
```

执行后的输出结果如下：

```
Beginning outer try block, no exceptions yet
Beginning inner try block, no exceptions yet
Exception caught, beginning inner catch block
Error type: ReferenceError
Error message: undefinedVariable is not defined
Executing inner finally block
Exception caught, beginning outer catch block
Error type: ReferenceError
Error message: undefinedVariable is not defined
Executing outer finally block
```

嵌套例外处理的好处在于使我们能够很好地分阶段处理错误，内部例外处理可以负责解决由错误引发的脚本代码问题，外部例外处理则用于负责提供给用户的反馈信息或者对例外信息进行日志记录。

九、结语

本文详细讨论了JavaScript语言的一个很重要的特征“例外处理”，Web开

发人员应该很好地掌握它并在实际应用中灵活处理，从而使包含脚本代码的HTML页面真正地不出例外、善解人意。