Javascript异常处理

> 在ES3之前js代码执行的过程中，一旦出现错误，整个js代码都会停止执行，这样就显的代码非常的不健壮。
>
> 在Java或C#等一些高级语言中，都提供了异常处理机制，可以处理出现的异常，而不会停止整个应用程序。
>
> 从ES3开始，js也提供了类似的异常处理机制，从而让js代码变的更健壮，及时执行的过程中出现了异常，也可以让程序具有了一部分的异常恢复能力。

一、Javascript的异常捕获机制

1.1 基本的try...catch语句

> ES3开始引入了 try-catch 语句，是 JavaScript 中处理异常的标准方式。
>
> **语法：**
>
> `try{`
> //可能发生异常的代码
> `}catch(error){`
> //发生错误执行的代码
> `}`
>
> 看下面的代码：

```
<script>
    try{
        console.log(b);
        console.log("我不会输出的，不要找了")

    }catch(error){
        console.log("发生错误了")
    }
    console.log("我try catch后面的代码")
</script>
```

```
⚠ ▶ Calling Element.createShadowRoot() for an element which already hosts a shadow root is deprecated. See https://chromestatus.com/features/4668884095336448 for more details.
发生错误了
我try catch后面的代码
```

> 说明：

1. 把有可能出的问题的代码放在 try 语句中。try语句中可以理论上可以写任何的代码，只要有一行代码出现问题，整个程序的执行流程就会立即调到catch语句中执行。

2. 一旦try中有一行代码发生异常，则这行出错代码的后面的try中的其他语句都不会再执行。比如上面代码中的`console.log(b);`这行代码会出错，则立即去执行catch中的代码。所以`console.log("我不会输出的，不要找了")`这行代码则不会再执行
3. 在执行catch中的代码之前，js引擎会首先根据错误类型自动创建一个错误，并通过catch后面的参数传递到catch中。不同的浏览器创建的error对象不一样，但是同创他们都包含一个message属性，值是这个错误的一些信息。
4. catch中的代码执行完毕之后，会继续执行后面的代码，程序不会停止下来。

1.2 finally语句

> 在 try...catch 中，try 中一旦出现错误则其他语句不能执行，如果不出现错误则 catch 中的语句不会执行。
>
> Javascript 参考其他编程语言，也提供了一种 finally 语句：不管 try 中的语句有没有错误，在最后都会执行 finally 中的语句。
>
> 即：try 中语句不发生错误执行完毕后会执行 finally 中的语句，try 中的语句发生错误，则执行catch中的语句，catch 中的语句执行完毕后也会执行 finally 中的语句。
>
> **语法：**
>
> `try{`
>
> `}catch(error){`
>
> `}finally{`
>
> `}`

```
<script>
    try{
        console.log(b);
        console.log("我不会输出的，不要找了")

    }catch(error){
        console.log("发生错误了")
    }finally {
        console.log("不管发生不发生错误，我都会执行")
    }
    console.log("我try catch后面的代码")
</script>
```

```
⚠ ▶ Calling Element.createShadowRoot() for an element which already hosts a shadow root is deprecated. See h
ures/4668884095336448 for more details.
发生错误了
不管发生不发生错误，我都会执行
我try catch后面的代码
```

> 所以在 finally 中我们可以放置我们必须要执行的代码。

注意：

1. 在js中，如果添加了 finally 语句，则 catch 语句可以省略。所以下面的代码也是正确的。
2. 如果没有 catch 语句，则一旦发生错误就无法捕获这个错误，所以在执行完 finally 中的语句后，程序就会立即停止了。
3. 所以，在实际使用中，最好一直带着 catch 语句。

```
<script>
    try{
        console.log(b);
        console.log("我不会输出的，不要找了")

    }finally {
        console.log("不管发生不发生错误，我都会执行")
    }
    console.log("我try catch后面的代码")
</script>
```

```
▶ Calling Element.createShadowRoot() for an element which already hosts a shadow root is deprecated. See https://w
ures/4668884095336448 for more details.
不管发生不发生错误，我都会执行
▶ Uncaught ReferenceError: b is not defined
    at test.html?_ijt=1g3gdc82emap2l8kpmfphne6vf:10
>
```

1.3 js中的错误类型

> 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。js共定义了下列 7 种错误类型：

- Error
- EvalError
- RangeError
- ReferenceError
- SyntaxError
- TypeError
- URIError

说明：

1. Error类型是基本的错误类型，其他类型都继承自这个类型。
2. EvalError 类型的错误会在使用 eval()函数而发生异常时被抛出
3. TypeError 类型在 JavaScript 中会经常用到，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
4. 一般情况，不同的错误，处理方式不一样。可以参考下面的处理方式。不过在实际开发中，很多程序员并没有形成处理错误的习惯。

```
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 {
      //处理其他的错误
  }
}
```

1.4 合理使用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，浏览器就不再报告错误了。这也是最简单的一种情况。

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 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，一有什么错误发生，也好可以恰当地处理它们。

在明明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，那么就应该先检查参数的类型，然后再决定 如何去做。在这种情况下，不应用使用 try-catch 语句。因为try...catch语句比较是比较好资源的事情。

二、throw主动抛出异常

2.1 抛出js内置错误类型的对象

> 在大部分的代码执行过程中，都是出现错误的时候，由浏览器(javascript引擎)抛出异常，然后程序或者停止执行，或被try...catch 捕获。
>
> 然而有时候我们在检测到一些不合理的情况发生的时候也可以主动抛出错误。
>
> 使用 throw 关键字抛出来主动抛出异常。

```
<script>
    throw new Error("你好坏");
    console.log("执行不到这里的")
</script>
```

```
▶ Calling Element.createShadowRoot() for an element which already hosts a shadow root   chrome-extension://g
is deprecated. See https://www.chromestatus.com/features/4668884095336448 for more details.
Uncaught Error: 你好坏
    at test.html:9
```

> 注意：

1. thow后面就是我们要抛出的异常对象。在以前的时候都是出现错误的时候浏览器抛出异常对象，

只是现在是我们自己主动抛出的异常对象。

2. 只要有异常对象抛出，不管是浏览器抛出的，还是代码主动抛出，都会让程序停止执行。如果想让程序继续执行，则有也可以用try...catch来捕获。
3. 每一个错误类型都可以传入一个参数，表示实际的错误信息。
4. 我们可以在适当的时候抛出任何我们想抛出的异常类型。`throw new SyntaxError("语法错误...");`

> 看下面的代码:

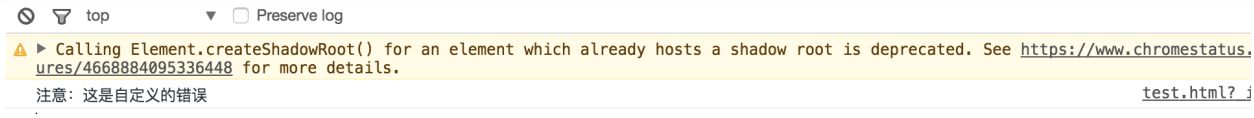
```
<script>
    /*该函数接收一个数字，返回他的平方。*/
    function foo(num) {
        if(typeof num == "number"){
            return num * num;
        }else{
            throw new TypeError("类型错误，你应该传入一个数字...")
        }
    }
    console.log(foo(33))
    console.log(foo("abc"))
</script>
```

```
▶ Calling Element.createShadowRoot() for an element which already hosts a shadow root is deprecated. See https://www.chromestatus.com/features/4668884095336448 for more details.
1089                                                                                                                                    test.ht
▶ Uncaught TypeError: 类型错误，你应该传入一个数字...                                                                                    test.ht
    at foo (test.html?_ijt=j6dn7l4uhvdam1c1cq1s64kkgl:14)
    at test.html?_ijt=j6dn7l4uhvdam1c1cq1s64kkgl:18
```

2.2 抛出自定义类型的错误对象

> 我们不仅仅可以抛出js内置的错误类型的对象，也可以自定义错误类型，然后抛出自定义错误类型的对象。
>
> 如果要自定义错误类型，只需要继承任何一个自定义错误类型都可以。一般直接继承Error即可。

```
<script>
    function MyError(message) {
        this.message = "注意：这是自定义的错误"
        this.name = "自定义错误";
    }
    MyError.prototype = new Error();
    try {
        throw new MyError("注意：这是自定义错误类型")
    }catch (error){
        console.log(error.message)
    }
</script>
```


top
Preserve log
▶ Calling Element.createShadowRoot() for an element which already hosts a shadow root is deprecated. See https://www.chromestatus.
ures/4668884095336448 for more details.
注意：这是自定义的错误
test.html?_j