JAVA认证:如何更合理的利用Java中的异常抛出Java认证考试
PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/559/2021_2022_JAVA_E8_AE_A4_E8_AF_81_c104_559607.htm 得编程语言中最让人不解的部分就是它能够创建错误。当时对Java语言中的throw关键字的第一反应就是“ 啊，这也太傻了，为什么我们想要引发一个错误(error)？ ”我觉得错误是我的敌人，应当避免的，所以创建错误是毫无用处甚至是危险的。我认为在JavaScript中加入这样的关键字是多此一举。但随着我编程经验的丰富，我逐渐变成了throw我的error粉丝。合理的使用它们会让对代码的调试和维护大大简化。 在编程的时候，Error通常出现在不期望的事情发生时。可能是传入函数的参数值不正确，或者是运算符的操作数不合法。为此编程语言定义了一个基本的规则：当上述情况发生时，就产生一个错误来让编程人员对代码进行修复。如果这些错误不被抛出或反馈给你，那么调试程序几乎是不可能的。如果所有的错误都“ 悄悄地 ”发生，那么你很难在第一时间发现问题所在，并将其修复。因此Error是开发者的朋友，而不是敌人。 Error的问题所在是它们会在错误的时间和错误的地点发生。更糟的是，默认的错误信息通常晦涩难懂，很难解释哪里出了问题。JavaScirpt的错误信息更是不包含任何有价值的信息，而且还很隐蔽(尤其是在IE里运行时)。想象一下如果能有这样的错误提示出现“ 因为某件事情发生导致某个函数调用失败 ”，那么立刻我们的调试任务就变得简单了，这就是throw自己的error的好处。我们可以把error想象成内嵌的异常类。在代码的某个特定的地点估计异常的发生肯定要比在所有的地方等待异常的发生

要简单。这不光在代码编写中，在产品设计中也是一个普遍认同的原则。就像在轿车上设计了挤压区域和框架，以便在受到撞击时会以期望的方式发生变形。因为知道了框架在受到撞击时会如何变形，哪些零件会失效，这样制造商就可以造出保证乘客安全的汽车。我们的代码也可以按照这样的思想编写。 虽然最近几年JavaScript有了很多进步，但是相比于其它语言的开发者，JavaScript开发者仍然只有少得可怜的调试工具。因此在 JavaScript中throw error就显得比其它语言更有价值。我们可以用throw关键字来抛出一个对象。我们可以抛出任何类型的对象，不过Error对象是最常用的： throw new Error("Something bad happened.") 当我们用这样的方式抛出错误，而这个错误又不被try-catch捕获时，浏览器就会用其通常的方式显示上面的错误信息(Something bad happened)。在IE里会在浏览器的左下角出现一个小图标，当双击图标时会弹出一个带着上面错误提示的对话框；安装有Firebug插件的火狐浏览器会在控制台显示错误信息；Safar和Chrome会在Web Inspector中显示；Opera会在错误控制台显示。一句话，它们会像你没有抛出错误时一样处理。但不同的是它会通过浏览器向你提供具体的信息，而不是一个发生错误的行列号。你可以为错误信息加入任何需要的信息，来帮你成功解决问题。我建议在错误信息中提供发生错误的函数名称以及错误原因。看下面这个函数： function addClass(element, className){element.className = " " className.} 这个函数的功能是向一个给定的element加入新的CSS class(这在JavaScript中非常普遍)。但如果element是null的时候会发生什么？你会得到一个这样的错误提示“ object expected ”，很隐晦。然后你需

要查看执行堆栈(如果浏览器支持这个功能)来准确定位错误的源头。如果我们抛出一个错误调试就变得简单了： function addClass(element, className){if (element != null amp. typeof element.className == "string"){element.className = " " className.} else {throw new Error("addClass(): First arg must be a DOM element.").}} 先不讨论如何精确的判断对象是否是一个DOM element，这个方法现在能够在非法的element参数传入时提供一个更明确的错误信息。看到了如此详尽的错误描述你就能立刻找到错误的源头了。我习惯把throw error看作是贴一个任务贴纸，告诉我错误的原因。 懂得了如何throw error只是事情的一半；懂得何时throw error则是另一半。因为JavaScript并不对参数进行类型检查，许多开发者都错误的认为他们应该在所有的函数中进行该检查。那样的话是不实际的，而且会降低脚本的执行效率。问题的关键在于找到最有可能出错的代码部分，并且只在那里throw error.一句话就是只在已经发生error的地方throw error. 如果一个函数只被一个已知的实体调用，那么错误检查基本上是没有必要的（例如私有函数就是这样）；如果你不能事先确定所有函数被调用的地点，那么你需要进行错误检查并throw自己的error.throw error最好的地方是功能函数，那些是脚本环境基本组成部分的，而且可以在任意地点被调用的函数。JavaScript的库函数就是这样的例子。 所有JavaScript的库函数都应当为已知的错误条件从它们的公共接口throw error.对于YUI，jQuery以及Dojo等等，我们无法确定会在何时何处调用它们的库函数。所以当你犯错时对你进行提示就是这些库函数的任务。为什么呢？因为你不可能到库函数内部去找出

错误所在。error的调用堆栈应当终止于库函数接口，不要再深入。没有什么比在12层函数嵌套中寻找错误更遭的事了；库函数开发人员有责任预防这种事情的发生。 这一条同样适用于私有的JavaScript库函数。许多Web应用程序都有它们自己专属的JavaScript库，可能是通过这些库来构建的，也可能是用库来代替公共的操作。库函数的作用是降低开发难度，这是通过向人们提供其抽象表达而不是复杂的实现细节来实现的。throw error可以让这些复杂的实现隐藏在安全的地方不被开发者发现。 JavaScript同样提供了try-catch语句，用来在浏览器处理之前捕获被throw的error.开发者常常会为到底是仅仅throw error还是用try-catch将其捕获而犹豫不决。我们应当只在程序栈的最底层throw error，就像前面提到的，最典型的就是JavaScript库函数。所有应用程序都应当在逻辑上具有处理error的能力，因此应当在底层模块中捕获 error. 在应用程序逻辑中我们总是知道为什么要调用某个函数，因此它们非常适合处理error.有一点要引起注意，就是永远不要在try-catch结构中使用空的catch语句；你应当用某种方法处理错误。这钟处理在开发中和最终生产时会有些不同，但必须进行处理。当错误发生时，不应当仅仅将其包裹在try- catch里不管这是掩盖错误而不是解决错误。 在JavaScript中throw error是一门艺术。在代码中找到适当的throw error的地点会花费一些时间。不过一旦你找到了这些地点，你的调试时间就会大大降低，而你对代码的满意度会获得提升。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com