异常和错误处理（基于Delphi_VCL）计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022__E5_BC_82_E5_B8_B8_E5_92_8C_E9_c97_645003.htm 有人在看了我的“ 如何将界面代码和功能代码分离（基于Delphi/VCL） ”之后，提到一个问题，就是如何对服务端的类的错误进行处理。 在基于函数的结构中，我们一般使用函数返回值来标明函数是否成功执行，并给出错误类型等信息。于是就会有如下形式的代码： RetVal := SomeFunctionToOpenFile(). if RetVal = E_SUCCESSED then ..... else if RetVal = E_FILENOTFOUND then ..... else if RetVal = E_FILEFORMATERR then ..... else then ..... 使用返回错误代码的方法是非常普遍的，但是使用这样的方法存在2个问题： 1、造成冗长、繁杂的分支结构（大量的if或case语句），使得控制流程变得复杂 2、可能会有没有被处理的错误（函数调用者如果不判断返回值的话） 而异常是对于错误处理的面向对象的解决方案。它可以报告错误，但需要知道的是，并非由于错误而引发了异常，而仅仅是因为使用了raise。 在Object Pascal中，抛出异常使用的是raise保留字。在任何时候（即使没有错误发生），raise都将会导致异常的发生。 异常可以使得代码从异常发生处立刻返回，从而保护其下面的敏感代码不会得到执行。通过异常从函数返回和正常从函数返回（执行到函数末尾或执行了Exit）对于抛出异常的函数本身来说是没有什么区别的。区别在于调用者处，通过异常返回后，执行权会被调用者的try...except块所捕获（如果它们存在的话）。如果调用者处没有try...except块的话，将不会继续执行后续语句，而是返回更上层的调用者，直至

找到能够处理该异常的 try...except块。异常被处理后，将继续执行try...except块之后的语句，控制权就被留在了处理异常的这一层。当异常处理程序感觉对异常的处理还不够完整时，需要更上层调用者继续处理，可以重新抛出异常（使用简单的raise.即可）将控制权交给更上层调用者。 如果根本就没有预设try...except块，则最终异常会被最外层的封装整个程序的VCL的try...except块所捕获。 因此，不会有不被处理的异常，换句话说，也就是不会有不被处理的错误（虽然错误和异常并不能划等号）。这也是异常机制比使用返回错误代码方法的优越之处。另外，异常被抛出后，其控制流程的走向非常清晰明了，不会造成流程失去控制的情况。 举个例子说明异常的工作机制，假设我们要打开某种特定格式的文件： 先定义两个异常类（从Exception继承） EFileNotFound = class(Exception). EFileFormatErr = class(Exception). 假设Form1上有一个按纽，按下按纽即打开文件： procedure TForm1.Button1Click(Sender: TObject). begin try ToOpenFile(). except on EFileNotFound do ShowMessage(\Sorry, I can\t find the file\). on EFileFormatErr do ShowMessage(\Sorry, the file is not the one I want\). on E:Exception do ShowMessage(E.Message). end. end. 以及打开文件的功能函数： procedure ToOpenFile. var RetVal:Integer. begin //Some code to openfile RetVal := -1. //open failed if RetVal = 0 then //success Exit else if RetVal = -1 then Raise EFileNotFound.Create(\File not found\) else if RetVal = -2 then Raise EFileFormatErr.Create(\File format error\) else //other error Raise Exception.Create(\Unknown error\). end. 程序中 TForm1.Button1Click 调用ToOpenFile，并预设了对ToOpenFile

可能抛出的异常处理的try...except。当然，也可以对 TForm1.Button1Click 的异常处理代码进行简化： procedure TForm1.Button1Click(Sender: TObject). begin try ToOpenFile(). except ShowMessage(\Open file failed\). end. end. 使用异常解决了使用返回错误代码方法存在的问题，当然，使用异常也不是没有代价的。异常会增加程序的负担，因此滥用异常也是不可取的。写若干 try...except和写数以千计的try...except之间是有很大区别的。用Chalie Calverts的话来说就是：“ 在似乎有用的时候，就应该使用try...except块。但是要试着让自己对这种技术的热情不要太过头 ”。 另外，Object Pascal引入了独特的try...finally结构。前面我说过，通过异常从函数返回和正常从函数返回是没有什么区别的。因此，函数中的栈中的局部对象，会自动得到释放，而堆中的对象则不会。而然，Object Pascal的对象模型是基于引用的，其存在于堆中，而非栈中。因此，有时我们在通过异常从函数返回之前需要清理一些局域的对象资源。 try...finally正是解决这个问题的。 我改写了以上的 ToOpenFile 的代码，这次让ToOpenFile过程中使用了一些资源，并在异常发生后（或者不发生）从函数返回前都会释放这些资源： procedure ToOpenFile. var RetVal: Integer. Stream: TStream. begin //Some code to openfile Stream := TStream.Create. RetVal := -1. //open failed try if RetVal = 0 then //success Exit else if RetVal = -1 then Raise EFileNotFound.Create(\File not found\) else if RetVal = -2 then Raise EFileFormatErr.Create(\File format error\) else //other error Raise Exception.Create(\Unknown error\). finally Stream.Free. end. end. 单步执行以上代码，可以看出，即使当RetVal的值为 0 时

，执行Exit后，仍然会执行finally中的代码，然后再从函数返回。由此保证了局部资源的正确释放。 try...except和try...finally的用途和使用场合是不同的，而很多初学者会将它们混淆。以下是笔者的一些个人认识：try...except一般用于调用者处捕获所调用的函数所抛出的异常并进行处理。而try...finally一般用于抛出异常的函数本身进行一些资源清理工作。 面向对象编程提供了“ 异常 ”这种错误处理的方案。善而用之，会对我们的工作有好处，可以显著改善所编写代码的质量。