ASP技术的出错处理

Error-handling in ASP

黄天成　谭志江　任清珍　(武汉大学　430072)

摘要：本文较为全面的分析了ASP网络编程中存在的出错处理问题。对错误进行了分类，针对每一类错误举出实例给予解决，对ASP程序的健壮性及开发调试有重要意义。

关键词：ASP　错误对象　出错处理

1 引言

在应用程序开发过程初期，对可能的错误处理往往受到忽视，但当开发进入到后期调试阶段时，程序员往往会发现在冗长、复杂的程序中找寻错误是一件非常烦人的事情，其工作量甚至会超过程序开发编制阶段的工作量。在ASP网络编程中也经常会遇到相同的问题，因此有必要认真对待错误的处理。ASP技术中的出错处理相对于别的技术（如VC、VB等）因没有跳转语句，缺少了一定的灵活性，但正确、有效的利用ASP本身提供的出错处理机制，也能简单有效的处理所遇到的错误，提高程序的健壮性，降低调试难度。

在此我们从一种新的角度出发，按照出错对象的不同，把ASP技术的应用错误分为三类：AspServer错误，vbscript错误，ADO数据库查询错误，针对这三种错误分别进行分析，并提出了解决的方法。

2 AspServer错误

AspServer错误顾名思义就是ASP的服务器出错了，比如说页面400是请求无效、页面404.1是未授权：登录失败、页面500是内部服务器错误等等，它通过ASP定制的出错页面来显示。当IIS中"不可思议"的出现一个错误（例如404 Not Found）时，页面看起来像是从服务器返回给客户端的一个错误页面，但实际上并不是这样。它们是普通的HTML网页，在对一个错误进行相应时被下载并发送给客户端。这些错误网页作为IIS的缺省安装部分，可根据需要定制。在Windws2000的IIS5.0环境下，这些页面放在WinNt＼Help＼iishelp＼common目录中，可用文本编辑器进行修改。（在Internet服务管理器中用右键点选所选的服务器，选择属性中的自定义错误信息可修改错误类型与错误网页之间的映射关系及所影响的目录范围。）

在ASP3.0中Server对象有一个名为GetLastError的新方法。与Vbscript的Err对象不同，不能为查看是否出现了错误而随时调用该方法，只能在一个ASP定制的错误网页中使用。GetLastError方法要做的事情是提供更多的有关错误源和错误原因的信息。GetLastError方法创建并返回一个叫ASPError的新对象。这个对象具有一系列的属性，这些属性返回有关在GetLastError方法调用之前出现的最新错误的信息。

ASPError对象提供了九个属性来说明所出现的错误的性质和错误源，并返回错误代码。其中主要有ASPCode（整型。由ASP/IIS产生的错误号）、ASPDescription（字符串型。与ASP相关的错误的详细说明）、File（字符串型。错误出现时正在处理的文件的名称）、Line（整型。产生错误的文件中的行号）等等（详情请见参考文献中的内容）。

类型500:100错误，是指ASP载入包含语法错误的页面，或者出现一个运行期错误。当出现500:100错误时将执行500:100.asp页面。应用GetLastError方法和ASPError对象，就可以定制符合我们自己习惯的500-100.asp页面。

```
.....
<%
Response.Status = "500 Internal Server Error"
Set objASPError = Server.GetLastError
%>
<b>错误类型：</b><br>
```

ASP/IIS 错误号为：<%= objASPError.ASPCode %>

COM 错误号为：<%= objASPError.Number %>_

(ox<%=Hex(objASPError.Number)%>)

错误实际代码为：<%= server.htmlencode (objASPError.source) %>

错误源为：<%= objASPError.Category %>

错误地址为：<%= objASPError.File %> 文件的第<%= objASPError.Line %>_行

错误详细说明：<%= objASPError.Description %>

<form action="<%=Request.ServerVariables ("HTTP_REFERER")%>"_ method="post">

<input type="submit" name="cmdclick" value="">   返回出错前_的一页<p>

</form>

如有问题请联络管理员yestzj@163.com

.....

如果再结合具体的编程，利用 Session 对象我们还可以在这定制页面上显示更多有用信息，当然会降低这个实例的通用性。

3 VBScript 错误

VBScript的错误处理机制是一种分层处理的结构。在有多个嵌套的过程中，如果在一个过程中出现错误，而该过程并没有已激活的错误处理程序（无 "On Error Resume Next" 语句），那么就要检查调用带有错误过程的那个过程，以便找出已经激活但并不是活动的错误处理程序。这个检查将在调用栈中连续向上进行，直到找到一个已激活但并不是活动的错误处理程序，或者到达调用栈的顶部。如果到达了调用栈的顶部，该错误即被视为未捕获的错误。如果遇到一个已被激活但不是活动的错误处理程序，便执行该错误处理过程，然后在包含该处理程序的过程中继续执行。在这一错误处理机制中，"On Error Resume Next" 语句和 "Err" 对象扮演着重要的角色。

有了 "On Error Resume Next" 语句，则在错误发生时，VBScript 解释器会简单的忽略错误，继续执行下一条语句。这语句不影响程序流程，保持了程序的健壮性，但也因此牺牲了一定的编程灵活性（VBScript 中没有 On Err Goto 语句）。这也意味着 VBScript 不支持大多数应用程序所采用的错误处理集中化机制。VBScript分层处理结构的具体体现就在于 "On Error Resume Next" 语句是一条过程级的语句，它只在包含有 "On Err" 声明的过程中有效。当错误出现在较底层的过程中时，VBScript解释器像许多其他语言一样，向上层的调用提交错误知道遇到带有错误处理的过程为止。如果最终没有发现这样的高层过程，那么脚本中止执行并向用户显示解释器给出的错误结果。高层过程有 "On Error Resume Next" 语句则 "Resume Next" 法则生效，该高层过程的下一条语句开始执行，即从调用低层过程的语句后恢复执行。

因简单 "Resume" 带来的安全隐患正好由 "Err" 对象来弥补。"Err" 对象是 VBScript 的固有对象，这意味着不需要做任何工作就可以使用它。当错误发生时，不需要特别的声明就可通过调用 "Err" 对象方便的获得有用的错误信息。"Err" 对象的重要属性有：Number（错误码，每种错误都有它自己特殊的错误码）、Description（于错误码对应的错误描述）、Source（说明是什么导致错误）等等。通过使用 "Err" 对象的信息，我们能够检查错误是否发生，并进一步获得有关的信息，帮助你在调试中探测错误。检查错误是否发生很容易，如果 "Err.Number" 等于 0，那么就没有问题，其他任何值均代表发生了错误。

Err 对象还包含两个有意思的方法："Clear" 方法和 "Raise" 方法。"Raise" 方法产生一个错误，更准确地说是仿真一个错误。换句话说，我们可以用 Raise 模拟任何一种错误。例如，仿真类型转换错误可以使用语句：Err.Raise 13，它使得 VBScript 的反映跟真的遇到可以产生错误代码为 13 的语句时一模一样。"Raise" 方法不仅在开发调试阶段用来仿真错误，而且在分层的错误处理时，往往被用来使程序流程转向拥有有效错误处理的较高过程。

详细说就是，如果错误处理程序的错误范围不包括实际产生的错误，则在具有激活的错误处理程序的过程内会产生意想不到的错误。此时，过程可能会无休止地执行下去，特别当错误处理程序还执行Resume语句的话，情况更是如此。为防止这种情况发生，可以在程序内的 CaseElse 语句或者 ELse 语句中使用 Err.Raise 方法。这实际上是在错误处理程序内生成一个运行期错误，从而迫使 VBScript 通过调用列表继续搜索能够处理错误的处理程序。

要完全掌握VBScript的错误处理，另一个需要知道的技术是 "Clear" 方法。"Err" 对象只保存了最近一次错误的信息，并且错误发生时 VBScript 解释器将相应的信息填入 "Err" 对象后会永久保存它，这样有时候就无法发现同一种错误是否再次发生了。虽然在程序流程执行到使用了 "On Error Resume Next" 的函数或者过程的末尾或者又碰到了 "On Error Resume Next" 语句时，VBScript就会清空 "Err" 对象（Err.Number 为 0），但在同一个过程中你如果要有多处检查 Err.Number 的话就要用到 Err.Clear 方法了。Err.Clear 方法复位 "Err" 对象的所有域并将 Err.Number置为0。这样当你需要保证有一个干净的错误记录来开始程序时，只需简单的在代码中插入 "Err.Clear"。

下面就是一个简单的出错处理实例：

```
<%
If Err.Number<>0 Then
    Response.Write "错误号:" & Err. Number& "<br>"
    Response.Write "错误的对象:"& Err. Source & "第" &Err.line&_
    "行" & "<br>"
```

```
    Response.Write " 错误说明:"& Err. Description &_"<br>"
    Err.Clear
End if
%>
```

错误分层处理机制虽然最终能捕获错误，但它却给调试工作带来了混乱。因此最好每个过程都能有自己独立的错误处理程序。

4 ADO数据库查询错误

数据库操作是ASP应用中不可缺少的一部分。在处理数据存储时发生错误的可能性也是存在的：安全性问题、试图更新已被其他用户删除的记录等等问题很多，我们也要对它进行一定程度的错误控制。

ADO为我们提供了Errors集合，它包含有单个ADO命令的执行而引起的每一个错误的Error对象。之所以要用集合的形式是因为在一个命令的执行过程中可能会引起多个错误。

对于Errors集合有两个地方需要注意：①每次执行ADO命令，如果发生错误就清空错误集，代之以新的错误内容。如果没有错误发生着Errors集合不受影响，也就是说，即使ADO命令成功执行，Errors集合也可能含有错误信息。②当包含信息的消息或警告被装入Errors集合时Err的错误号为0。所以不能只检查集合中的错误号而假定错误已发生。而是以Errors.count中的个数来判断是否出错，大于零自然是有错误发生。

Errors集合只是Connection对象的一个属性，只能通过Connection对象访问。Recordset对象有一个ActiveConnection属性，含有当前记录集的Connection对象，使用RS.ActiveConnection.Errors即可得到Errors集合。要想查看Errors集合中的每一个Err只需要用下列循环语句：

```
For Each Err in RS.ActiveConnect.Errors
    'Display Error
    ......
Next
```

Error对象包含的属性有：Number(ADO错误号)，NativeError(从数据提供者获得的错误号)，SQLState(连接到SQL数据库时，5位的SQL状态代码),Source(引起错误的对象),Description(错误说明文本)。

对于数据库操作我们最好在每一行ADO代码后检查Errors集合，所以编写一个显示出错信息的子程序会带来很多方便。

```
Function CheckErr(objConn)
    Dim objError
    If objConn.Errors.Count > 0 then
        CheckErr = True
        For Each objError in objConn .Errors
            Response.Write "ADO 错误号:
            "& objError. Number&"<br>"
            Response.Write "ODBC错误号:
            "& objError. NativeError &_
            "<br>"
            Response.Write "SQL状态代码:
            "& objError. SQLState &_
            "<br>"
            Response.Write " 错误的对象:
            "& objError. Source &"<br>"
            Response.Write " 错误说明:"&
            objError. Description &_
            "<br>"
            next
    Else
        CheckErr = False
    End if
End Function
```

然后我们在每次ADO数据库操作之后加上下列检查语句：

```
On Error Resume Next
......
RS.open' 类似的数据库操作语句
If CheckErr(RS.ActiveConnection) = False then
    ' 对RS进行你需要的操作
    ......
End if
```

就可对ADO错误进行检测并处理。

5 值得注意的两点

在把错误分类并进行详细分析之后，我们要注意的是在实际中错误并不会总是单独出现，而往往是几个错误连着出现，这就要求我们在检测错误的时候能把上述的几类方法综合起来使用。例如，在进行ADO数据库操作之前VBScript出现错误，程序"Resume"后即使数据库操作成功，数据也会出错。所以应该在数据库操作之前进行VBScript的出错检验，在数据库操作之后进行ADO的出错检验。

另外，因为ASP难以做到错误的集中处理，在检验点的选择上就要求细致入微，在一些诸如页面跳转(Redirect)、表单提交、Session变量使用等方面都要提高警惕。编制适合自己的检验子程序，宁滥勿缺的使用它，相信会给你带来好处。

6 总结

本文所讨论的问题可能是大多数编程人员所不喜欢又不得不做的工作，这就使我们迫切的期望能有一种规范的错误处理机制来减少莫名错误的出现时带来的困扰，通过分析可能出现的不同种类的错误，弄清楚缺省的ASP和脚本引擎错误处理系统捕获错误的机制，有助于我们更容易的跟踪错误、处理错误。面对复杂多变的出错情况，恒心和毅力是必不可少的，而本文提供的几个实例相信也将有助于ASP编程中的错误处理。

参考文献

1 [美] Bill Schongar 等，《VBScript揭秘》，电子工业出版社，1998。

2 [美] Richard Anderson、Chris Blexrud 等，《ASP3高级编程》，机械工业出版社，2000。