文章
姚 鑫 · 二月 9, 2021 阅读大约需 分钟

第二十九章 Caché 变量大全 $ZERROR 变量

第二十九章 Caché 变量大全 $ZERROR 变量

包含上一个错误的名称和位置。

大纲

```
$ZERROR
$ZE
```

描述

$ZERROR包含最新错误的名称，最新错误的位置（在适用的情况下）以及（对于某些错误代码而言）有关导致错误的原因的其他信息。$ZERROR始终包含相应语言模式的最新错误。

$ZERROR值旨在错误后立即使用。由于$ZERROR值可能不会在例程调用中保留，因此**建议保留**$ZERROR值以供以后使用的用户应将其复制到变量中。 **强烈建议用户在使用后立即将$ZERROR设置为空字符串("")。**

$ZERROR中包含的字符串可以是以下任何一种形式：

```
<error>
<error>entryref
<error> info
<error>entryref info
```

- <error> 错误名称。错误名称始终以全部大写字母返回，并用尖括号括起来。它可能包含空格。
- entryref 对发生错误的代码行的引用。它由标签名称和距该标签的行偏移量组成，后跟^和程序名称。此entryref紧跟在错误名称的右尖括号之后。从终端调用$ZERROR时，此entryref信息没有意义，因此不会返回。对最近使用ZLOAD加载到例程缓冲区中的例程的引用。
- info 特定于某些错误类型的附加信息（见下表）。此信息与<error>或<error>entryref之间用空格分隔。如果有多个组件要提供信息，则用逗号分隔。

例如，一个程序（名为zerrortest）包含以下例程（名为ZerrorMain），该例程试图写入fred（一个未定义的局部变量）的内容：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR()
ClassMethod ZERROR()
{
ZerrorMain
    TRY {
    SET $ZERROR=""
```

```
    WRITE "$ZERROR = ",$ZERROR,!
    WRITE fred }
    CATCH {
    WRITE "$ZERROR = ",$ZCVT($ZERROR,"O","HTML")
    }
}
```

```
DHC-APP> d ##class(PHA.TEST.SpecialVariables).ZERROR()
$ZERROR =
$ZERROR = &lt;UNDEFINED&gt;zZERROR+5^PHA.TEST.SpecialVariables.1 *fred
```

在上面的示例中，第一个$ZERROR包含一个空字符串(“”)，因为自从$ZERROR重置为空字符串以来没有发生任何错误。尝试写入未定义的变量会设置$ZERROR并将其抛给CATCH块。此$ZERROR包含ZerrorMain+4^zerrortest*fred，指定错误的名称、位置和特定于该类型错误的附加信息。在本例中，附加信息是未定义的局部变量fred的名称；星号前缀表示它是局部变量。(请 注 意，本例中使用$ZCVT($ZERROR, “O”, “HTML”)，因为Caché错误名称用尖括号括起来，并且本例从Web浏览器运行。)

Entryref可能如下所示：

- ZerrorMain+4^zerrortest--程序zerrortest中标签ZerrorMain的4行偏移量
- ZerrorMain^zerrortest--在程序zerrortest中没有与标签ZerrorMain的偏移量；标签行中出现错误
- +3^zerrortest--从程序zerrortest开始的3行偏移量；错误行前面没有标签

$ZERROR值的最大长度为512个字符。超过该长度的值将被截断为512个字符。

AsSystemError() Method

%Exception.SystemException类的AsSystemError()方法返回与$ZERROR相同的值。下面的示例显示了这一点：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR1()
ClassMethod ZERROR1()
{
    TRY {
        KILL mylocal
        WRITE mylocal
    }
    CATCH myerr {
        WRITE "AsSystemError is: ",myerr.AsSystemError(),!
        WRITE "$ZERROR is:       ",$ZERROR
    }
}
```

```
DHC-APP>d ##class(PHA.TEST.SpecialVariables).ZERROR1()
AsSystemError is: <UNDEFINED>zZERROR1+3^PHA.TEST.SpecialVariables.1 *mylocal
$ZERROR is:       <UNDEFINED>zZERROR1+3^PHA.TEST.SpecialVariables.1 *mylocal
```

在Try/Catch异常处理块结构中，AsSystemError()比$ZERROR更可取，因为$ZERROR可能会被异常处理期间发生的错误覆盖。

有关某些错误的其他信息

当发生某些类型的错误时，$ZERROR将以以下格式返回错误：

```
<ERROrCODE>entryref info
```

INFO组件包含有关错误原因的附加信息。下表列出了错误列表，其中包括附加信息和该信息的格式。错误代码与INFO组件之间用空格字符分隔。

错误代码
<UNDEFINED>
<SUBSCRIPT>
<NOROUTINE>
<CLASS DOES NOT EXIST>
<PROPERTY DOES NOT EXIST>
<METHOD DOES NOT EXIST>
<PROTECT>
<THROW>
<COMMAND>
<DIRECTORY>
<FRAMESTACK>

例程(或方法)本地变量的名称以及未定义例程、类、属性或方法的名称都以星号(*)为前缀。进程-

专用全局变量由其^||前缀标识。全局变量由它们的^(插入符号)前缀标识。类名以其%前缀形式表示。

以下示例显示了指定错误原因的其他错误信息。在每种情况下，指定的项都不存在。请 注 意，生成错误的INFO组件与错误名称之间用空格分隔。星号(*)表示局部变量、类、属性或方法。插入符号(^)表示全局，^||表示进程私有全局。

<unfined>错误示例：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR2()
ClassMethod ZERROR2()
{
UndefTest ;
    SET $NAMESPACE="SAMPLES"
    KILL x,abc(2)
    KILL ^xyz(1,1),^|"USER"|xyz(1,2)
    KILL ^||ppg(1),^||ppg(2)
    TRY {
        WRITE x
    }             // ????????
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        WRITE abc(2)
    }         // ??????????
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        WRITE ^xyz(1,1)
    }          // ????????
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        WRITE ^|"USER"|xyz(1,2)
    }  // ???????????????
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        WRITE ^||ppg(1)
    }     // ???????????
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        WRITE ^|"^"|ppg(2)
    }  // ???????????
    CATCH {
        WRITE $ZERROR,!
    }
}
```

```
DHC-APP>d ##class(PHA.TEST.SpecialVariables).ZERROR2()
<UNDEFINED>zZERROR2+7^PHA.TEST.SpecialVariables.1 *x
<UNDEFINED>zZERROR2+13^PHA.TEST.SpecialVariables.1 *abc(2)
```

```
<UNDEFINED>zZERROR2+19^PHA.TEST.SpecialVariables.1 ^xyz(1,1)
<UNDEFINED>zZERROR2+25^PHA.TEST.SpecialVariables.1 ^xyz(1,2)
<UNDEFINED>zZERROR2+31^PHA.TEST.SpecialVariables.1 ^||ppg(1)
<UNDEFINED>zZERROR2+37^PHA.TEST.SpecialVariables.1 ^||ppg(2)
```

<SUBSCRIPT>错误示例：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR3()
ClassMethod ZERROR3()
{
SubscriptTest ;
    DO $SYSTEM.Process.NullSubscripts(0)
    KILL abc,xyz
    TRY {
        SET abc(1,2,3,"")=123
    }
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        SET xyz(1,$JUSTIFY(1,1000))=1
    }
    CATCH {
        WRITE $ZERROR,!
    }
}
```

```
DHC-APP>d ##class(PHA.TEST.SpecialVariables).ZERROR3()
<SUBSCRIPT>zZERROR3+5^PHA.TEST.SpecialVariables.1 *abc() Subscript 4 is ""
<SUBSCRIPT>zZERROR3+11^PHA.TEST.SpecialVariables.1 *xyz() Subscript 2 > 511 chars
```

<NOROUTINE>错误示例：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR4()
ClassMethod ZERROR4()
{
NoRoutineTest ;
    KILL ^NotThere
    TRY {
        DO ^NotThere
    }
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        JOB ^NotThere
    }
    CATCH {
        WRITE $ZERROR,!
    }
    TRY {
        GOTO ^NotThere
    }
```

```
    CATCH {
        WRITE $ZERROR,!
    }
}


DHC-APP>d ##class(PHA.TEST.SpecialVariables).ZERROR4()
<NOROUTINE>zZERROR4+4^PHA.TEST.SpecialVariables.1 *NotThere
<NOROUTINE>zZERROR4+10^PHA.TEST.SpecialVariables.1 *NotThere
<NOROUTINE>zZERROR4+16^PHA.TEST.SpecialVariables.1 *NotThere
```

对象错误的示例：

```
DHC-APP>DO $SYSTEM.SQL.MyMethod()

DO $SYSTEM.SQL.MyMethod()
^
<METHOD DOES NOT EXIST> *MyMethod,%SYSTEM.SQL
DHC-APP>WRITE $SYSTEM.XXQL.MyMethod()

WRITE $SYSTEM.XXQL.MyMethod()
^
<CLASS DOES NOT EXIST> *%SYSTEM.XXQL
DHC-APP>SET x=##class(%SQL.Statement).%New()

DHC-APP>WRITE x.MyProp

WRITE x.MyProp
^
<PROPERTY DOES NOT EXIST> *MyProp,%SQL.Statement
```

<PROTECT>错误的示例(在Windows上)：

```
    // ?????SYS?????????
    SET x=^|"%SYS"|var
    <PROTECT> ^var,c:\intersystems\cache\mgr\
```

调用用户定义函数时的<command>错误示例。在本例中，MyFunc Quit命令不返回值。这将生成个<command>错误，其中entryref指定$$MyFunc调用的位置，INFO消息指定QUIT命令的位置：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR5()
ClassMethod ZERROR5()
{
Main
    TRY {
        KILL x
        SET x=$$MyFunc(7,10)
        WRITE "returned value is ",x,!
        RETURN
    }
    CATCH {
        WRITE "$ZERROR = ",$ZCVT($ZERROR,"O","HTML"),!
    }
```

```
MyFunc(a,b)
   SET c=a+b
   QUIT
}
```

```
DHC-APP>d ##class(PHA.TEST.SpecialVariables).ZERROR5()
$ZERROR = &lt;COMMAND&gt;zZERROR5+4^PHA.TEST.SpecialVariables.1 *Function must return
 a value at zZERROR5+13^PHA.TEST.SpecialVariables.1
```

使用PUBLIC关键字将函数作为过程调用时，出现相同的<command>错误：

```
Main
    TRY {
        KILL x
        SET x=$$MyFunc(7,10)
        WRITE "returned value is ",x,!
        RETURN
    }
    CATCH {
        WRITE "$ZERROR = ",$ZCVT($ZERROR,"O","HTML"),!
    }
MyFunc(a,b) PUBLIC {
   SET c=a+b
   QUIT
   }
```

<DIRECTORY>错误示例(在Windows上)：

```
/// d ##class(PHA.TEST.SpecialVariables).ZERROR6()
ClassMethod ZERROR6()
{
    TRY {
        SET prev=$SYSTEM.Process.CurrentDirectory("bogusdir")
        WRITE "previous directory: ",prev,!
        RETURN
    }
    CATCH {
        WRITE "$ZERROR = ",$ZCVT($ZERROR,"O","HTML"),!
        QUIT
    }
}
```

```
DHC-APP>d ##class(PHA.TEST.SpecialVariables).ZERROR6()
$ZERROR = &lt;DIRECTORY&gt;zCurrentDirectory+2^%SYSTEM.Process.1 *e:\dthealth\db\dthi
s\data\bogusdir\
```

5.1版本之前的错误处理代码

在Caché5.1和后续版本的这些错误代码中添加INFO组件的结果是，假设$ZERROR中的字符串格式的5.1版本之前的错误处理例程可能需要重新设计才能像以前一样工作。例如，以下内容在5.1版中将不再有效：

```
WRITE "Error line: ", $PIECE($ZERROR, ">", 2)
```

并应更改为类似以下内容：

```
WRITE "Error line: ", $PIECE($PIECE($ZERROR, ">", 2), " ", 1)
```

注意

ZLOAD和错误消息

在ZLOAD操作之后，加载到例程缓冲区中的例程的名称出现在后续错误消息的entryref部分。这将在整个过程中持续存在，或者直到使用ZREMOVE删除，或者被另一个ZLOAD删除或替换。以下终端示例显示例程缓冲区内容的此显示：

```
SAMPLES>ZLOAD Sample.Person.1
SAMPLES>WRITE 6/0
<DIVIDE>^Sample.Person.1
SAMPLES>WRITE fred
<UNDEFINED>^Sample.Person.1 *fred
SAMPLES>WRITE ^fred
<UNDEFINED>^Sample.Person.1 ^fred
SAMPLES>ZNAME "USER"
USER>WRITE 7/0
<DIVIDE>^Sample.Person.1
USER>ZREMOVE
USER>WRITE ^fred
<UNDEFINED> ^fred
```

$ZERROR和程序栈

$ZERROR字符串的<error>部分包含最新的错误消息。$ZERROR字符串的entryref部分的内容反映了最近错误的堆栈级别。以下终端会话试图调用无意义的命令gobbledegook，导致<SYNTAX>错误。它还运行ZerrorMain(上面指定)，产生$ZERROR值<unfined>。此终端会话期间的后续$ZERROR值反映了此程序调用，如下所示：

```
SAMPLES>gobbledegook
SAMPLES>WRITE $ZERROR
<SYNTAX>
SAMPLES>DO ^zerrortest
SAMPLES>WRITE $ZERROR
<UNDEFINED>ZerrorMain+2^zerrortest *FRED
SAMPLES 2d0>gobbledegook
SAMPLES 2d0>WRITE $ZERROR
<SYNTAX>^zerrortest
SAMPLES 2d0>QUIT
SAMPLES>WRITE $ZERROR
<SYNTAX>^zerrortest
SAMPLES>gobbledegook
SAMPLES>WRITE $ZERROR
<SYNTAX>
```

设置$ZTRAP时的$ZERROR操

发生错误并设置$ZTRAP时，Caché在$ZERROR中返回错误消息，并分支到为$ZTRAP指定的错误陷阱处理程序

设置$ZERROR

只有在Caché模式下，才能使用set命令将$ZERROR设置为最多512个字符的值。长度超过512个字符的值将被截断为512。

强烈建议在错误处理后将$ZERROR重置为空字符串("")。

#Caché #InterSystems IRIS #InterSystems IRIS for Health

源 URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E4%BA%8C%E5%8D%81%E4%B9%9D%E7%AB%A0-cach%C3%A9-%E5%8F%98%E9%87%8F%E5%A4%A7%E5%85%A8-zerror-%E5%8F%98%E9%87%8F