Extensiones de lenguaje

Artículo
Kurro Lopez · Ago 1, 2019 Lectura de 7 min

Extensiones de lenguaje

¡Hola a tod@s!

En este artículo trataré una característica particular de Caché, muy útil, pero que probablemente no se conoce ni se usa bien. Me refiero a la función de Extensiones de lenguaje.

Esta característica permite extender los comandos, variables especiales y funciones disponibles en Caché Object Script con comandos, variables especiales y funciones propias. Esta funcionalidad también se aplica a otros idiomas que Caché admite en el servidor, incluidos Caché Basic y Multivalue Basic.

¿Por qué necesitaría o querría agregar nuevos comandos?

Hay varios casos de uso. Por lo general, me gusta usar esto para ahorrar tiempo durante el desarrollo y la depuración, al finalizar las llamadas a funciones de uso frecuente, o una serie de comandos, o simplemente una llamada a funciones largas, en un solo comando personalizado abreviado.

Veamos un ejemplo. Los desarrolladores que depuran en la línea de comando regularmente necesitan recuperar el texto de error del último error generado, representado por la variable de objeto de estado (%objlasterror), que se definiría en su partición. Así es como lo haces:

```
> Do ##class(%SYSTEM.Status).DisplayError(%objlasterror)
```

En la llamada anterior, invocamos el método DisplayError de la clase %SYSTEM.Status, pasando en %objlasterror. Esta es una cadena larga para escribir con frecuencia, invitando errores tipográficos que causan demoras. Lo que podemos hacer es introducir el comando breve de nuestro propio sitio, que haría lo mismo. Nuestro nuevo comando podría diseñarse para tomar opcionalmente cualquier variable de objeto de estado como argumento, y si se deja fuera, se supone que %objlasterror es el argumento.

Aquí está el nuevo comando que he llamado "ZOE", en acción:

```
SAMPLES>set p=##class(Sample.Person).%New()
SAMPLES>do p.%Save()    ; (save will fail, and define %objlasterror).
SAMPLES>zoe
ERROR #7209: Datatype value '' does not match PATTERN '3N1"-"2N1"-"4N'
```

SAMPLES>set tSC= ##class(%SYSTEM.Status).Error(5001,”Sample Status Text”)
SAMPLES>zoe tSC
ERROR #5001: Sample Status Text

Entonces, ¿cómo se implementan los comandos personalizados, variables especiales y funciones?

Las extensiones se implementan creando una rutina Caché Objectscript llamada específicamente %ZLANGCnn, %ZLANGVnn o %ZLANGFnn, que define la lógica detrás de nuevos comandos, variables especiales y funciones

respectivamente.

El nn en los nombres de rutina anteriores, denotan los idiomas en los que estas extensiones deben estar disponibles. Los valores comunes de nn incluyen:

00 - Caché Object Script
09 - Caché Basic
11 - Multivalued Basic

(Consulte el enlace de documentación al final del artículo para ver la lista completa)

He creado la rutina %ZLANGC00 para definir el comando ZOE anterior, y lo guardé en el namspace %SYS. Puedo agregar tantos comandos de extensión de idioma como desee dentro de la misma rutina (y esa rutina, por supuesto, puede invocar otro código).

Aquí hay una muestra de la rutina %ZLANGC00 que definí para este artículo:

```
%ZLANGC00 ; Language Extensions (Commands) - SP
          ;
          Quit    ; quit nicely if this routine is actually invoked directly.

          ; Command to display error text of status object supplied, OR of
%objlasterror
          ; if no argument defined.
ZOERROR(%err) do ZOE(.%err)  ; note the '.' so it works if %err is undefined.
ZOE(%err) ;
          if $get(%err)="",$get(%objlasterror)'="" {
               set %err=%objlasterror
          }
          if $get(%err)'="" do $System.OBJ.DisplayError(%err)
          quit

ZOLIST    ; Command to quickly output the list of defined objects in my partition
ZOL       ;
          do $System.OBJ.ShowObjects()
          quit
```

Observe también que en el ejemplo anterior, que también he introducido un alias largo (ZOERROR) para el comando ZOE que pasará y hará exactamente lo mismo. También hay un segundo comando, sin argumentos, ZOL (ZOLIST) para eliminar las variables de objeto definidas actualmente. Tenga en cuenta que cuando las implementaciones de código personalizado no toman argumentos (por ejemplo, ZOLIST, ZOL), el código puede pasar de una etiqueta de línea (ZOLIST) a la siguiente. Sin embargo, si se requieren argumentos (por ejemplo, ZOERROR y ZOE), entonces la función de alias larga debe llamar explícitamente a la función de alias corto.

Del mismo modo, puede crear sus propias funciones personalizadas. Aquí está la implementación de una función para devolver la r, en lugar de comandos. Considere, por ejemplo, una función para devolver el trimestre del año dada una fecha $Horolog.

```
%ZLANGF00 ; Language Extensions (Functions) - SP
          ;
          Quit     ; quit nicely if this routine is actually invoked directly.

          ; Function to return the quarter (Q1, Q2, etc..) of a given date supplied
in the
          ; internal date format ($HOROLOG).
          ;
ZYEARQ(hdate)    ;
          do ZYQ(.hdate)
```

```
ZYQ(hdate)      ;
        ;
        quit:+$get(hdate)'?5n ""   ; quit if argument passed in, isn't in the format

        set date=$zdate(hdate,1)   ; returned AS MM/DD/YY
        set month=+date            ; convert date string to numeric, returns month
alone.

        set quarter=$normalize((month+1)/3,0)
        quit "Q"_quarter
```

Nota: Desde el punto de vista de las mejores prácticas, las funciones como el ejemplo anterior probablemente estén más destinadas a ser utilizadas por el código de la aplicación (en lugar de un administrador o desarrollador que trabaje en el indicador de línea de comandos) e idealmente preferiría implementar lo anterior como un método en una clase de tipo de utilidad de una aplicación.

Uso:

```
SAMPLES>WRITE $ZYQ($horolog)
“Q2”
```

Finalmente, puede definir las variables especiales de su propio entorno. Las variables especiales son variables mantenidas por el sistema.

Puede recuperar y, en algunos casos, establecer su valor. Una variable especial común de solo lectura es, por ejemplo, $ZVERSION que devuelve una cadena que indica la versión instalada actualmente. Una variable especial que puede establecer es $NAMESPACE, que tiene el efecto de cambiar los espacios de nombres automáticamente.

Si implementa una variable especial, que registra un valor, usted es responsable de almacenar ese valor en algún lugar.

La siguiente es una implementación de una variable especial que devuelve el número de segundos transcurridos desde la medianoche

```
%ZLANGV00    ; Custom extended Special variables - SP
        ;
        Quit     ; quit nicely if this routine is actually invoked directly.

        ; Special variable representing seconds since midnight
ZHTIME() quit $$ZHT()
ZHT()   quit $piece($horolog,",",2)
```

Uso:

```
SAMPLES>Write !,"Good ",$select($zht>43200:"Afternoon.",1:"Morning.")
```

Conclusión y pensamientos finales:

Hay un par de cosas a tener en cuenta si tiene la intención de adoptar esta funcionalidad.

Creo que vale la pena presentar un puñado de comandos útiles y socializar su propuesta con su equipo de desarrollo para su aprobación y revisión, y finalmente, implementarla según sea necesario. Esto se debe a que una vez que implemente las rutinas %ZLANG* en su entorno, estarán disponibles para todos y en todos los

espacios de nombres.

Creo que un pequeño conjunto está garantizado, no se exceda, y definitivamente solo si no hay una alternativa nativa.

Mantenga los comandos cortos, con la versión larga (si la implementa) no más de 6-8 caracteres.

Las extensiones de lenguaje son inherentemente más lentas al llamar a comandos, funciones o variables nativas. Si bien puede invocar sus nuevas funciones en su aplicación, como parte de su aplicación, puede valer la pena designar su uso para la depuración o administración de la línea de comandos, eligiendo en su lugar usar los comandos nativos dentro del código de la aplicación.

Cuando usa su comando extendido, funciones o variables, se comportan como las nativas, lo que significa que no distinguen entre mayúsculas y minúsculas y obedecen otras estructuras, como las expresiones condicionales posteriores. Por ejemplo, escriba: X> 1 $ zht

Tenga cuidado de no definir extensiones de clientes que se superpongan con comandos nativos existentes, funciones o variables especiales, ya que su versión no funcionará.

Tenga cuidado al implementar su código personalizado, ya que no modifica el estado que esperaría que permanezca igual, después de que se ejecuta su código, por ejemplo, variables especiales como $ZREFERENCE (la referencia global actual) o $TEST (valor real resultante del último comando que usa la opción de timeout) no debería cambiar si ejecuta su comando personalizado.

La documentación sobre este tema se puede encontrar aquí:

http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSTU_customize#GSTU_customize_zlang

¡Espero que les haya resultado útil!

#Lenguajes #Consejos y trucos #Caché
10 1 0 0 122

Log in or sign up to continue
Añade la respuesta

URL de fuente: https://es.community.intersystems.com/post/extensiones-de-lenguaje