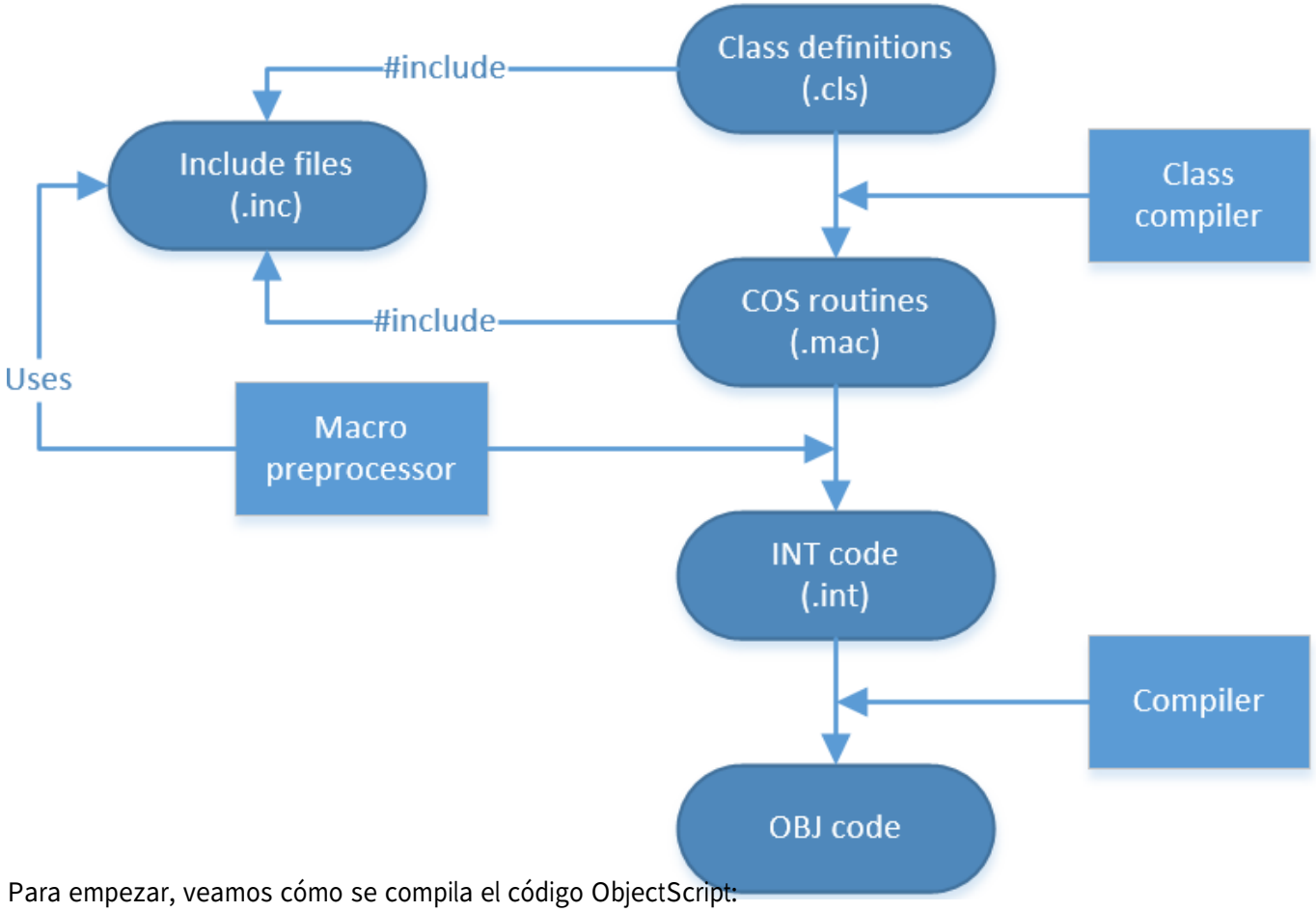
Artículo
Kurro Lopez · Mayo 17, 2020 Lectura de 7 min

Macros en InterSystems Caché

En este artículo me gustaría contarle acerca de las macros en InterSystems Caché. Una macro es un nombre simbólico que se reemplaza con un conjunto de instrucciones durante la compilación. Una macro puede "desplegarse" en varios conjuntos de instrucciones cada vez que se llama, dependiendo de los parámetros que se le pasen y los escenarios activados. Esto puede ser tanto código estático como el resultado de la ejecución de ObjectScript. Echemos un vistazo a cómo puede usarlos en su aplicación.

Compilación



Para empezar, veamos cómo se compila el código ObjectScript:

1. El compilador de clases usa definiciones de clase para generar código MAC
2. En algunos casos, el compilador usa clases como base para generar clases adicionales. Puede ver estas clases en el estudio, pero no debe cambiarlas. Esto sucede, por ejemplo, al generar clases que definen servicios web y clientes
3. El compilador de clases también genera un descriptor de clase utilizado por Caché en tiempo de ejecución
4. El preprocesador (también conocido como macro preprocesador, MPP) utiliza archivos INC y reemplaza las macros. Además, también procesa SQL incorporado en rutinas ObjectScript
5. Todos estos cambios tienen lugar en la memoria; el código del usuario permanece sin cambios
6. Después de eso, el compilador crea código INT para las rutinas de ObjectScript. Esta capa se conoce

como código intermedio. Todo el acceso a los datos en este nivel se proporciona a través de globals
7. El código INT es compacto y puede ser leído por un humano. Para verlo en el Studio, presione Ctrl + Shift + V.
8. El código INT se usa para generar código OBJ
9. El código OBJ es utilizado por la máquina virtual Caché. Una vez que se genera, el código CLS / MAC / INT ya no es necesario y puede eliminarse (por ejemplo, si queremos enviar un producto sin el código fuente)
10. Si la clase es persistent, el compilador SQL creará las tablas SQL correspondientes

Macros

Como mencioné antes, una macro es un nombre simbólico que se reemplaza por el preprocesador con un conjunto de instrucciones. Una macro se define con la ayuda del comando #Define seguido del nombre de la macro (quizás con una lista de argumentos) y su valor:

```
#Define Macro[(Args)] [Value]
```

¿ Dónde se pueden definir las macros? Ya sea en el código o en forma independiente enarchivos INC que contiene solo macros. Los archivos necesarios se incluyen en las clases al comienzo de las definiciones de clase utilizando el comando Include MacroFileName - este es el método principal y preferido para incluir macros en clases. Las macros incluidas de esta manera se pueden usar en cualquier parte de una clase. Puedes usar el comando #Include MacroFileName para incluir un archivo INC con macros en rutinas MAC o el código de métodos de clase particulares.

Tenga en cuenta que los generadores de métodos requieren #Include dentro de su propio cuerpo si quieres usar macros en tiempo de compilación o uso de la palabra clave IncludeGenerator en una clase.

Para hacer que la macro esté disponible en el autocompletado de estudio, agregue `///` en una línea anterior:

```
///
#Define Macro[(Args)] [Value]
```

Ejemplos

Ejemplo 1

Pasemos ahora a algunos ejemplos, y ¿ por qué no comenzamos con el mensaje estándar "Hello, World!"? Código COS:

```
Write "Hello, World!"
```

Crearemos una macro llamada HW que escribirá esta línea:

```
#define HW Write "Hello, World!"
```

Todo lo que necesitamos hacer ahora es escribir $$$HW ($$$ para llamar a la macro, luego su nombre):

```
ClassMethod Test()
{
     #define HW Write "Hello, World!"
     $$$HW
}
```

Se convertirá en el siguiente código INT durante la compilación:

```
zTest1() public {
     Write "Hello, World!" }
```

El siguiente texto se mostrará en la terminal cuando se llame a este método:

```
Hello, World!
```

Ejemplo 2

Usemos variables en el siguiente ejemplo:

```
ClassMethod Test2()
{
     #define WriteLn(%str,%cnt) For ##Unique(new)=1:1:%cnt { ##Continue
         Write %str,! ##Continue
     }

     $$$WriteLn("Hello, World!",5)
}
```

Aquí la cadena %str está escrita %cnt veces. Los nombres de las variables deben comenzar con %. El comando ##Unique(new) crea una nueva variable única en el código generado, mientras que el comando ##Continue nos permite continuar definiendo la macro en la siguiente línea. Este código se convierte en el siguiente código INT:

```
zTest2() public {
     For %mmmu1=1:1:5 {
         Write "Hello, World!",!
     } }
```

El terminal mostrará lo siguiente:

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Ejemplo 3

Pasemos a los ejemplos más complejos. El operador ForEach puede ser muy útil para iterar a través de globals, creémoslo:

```
ClassMethod Test3()
{
    #define ForEach(%key,%gn) Set ##Unique(new)=$name(%gn) ##Continue
    Set %key="" ##Continue
    For { ##Continue
        Set %key=$o(@##Unique(old)@(%key)) ##Continue
        Quit:%key=""
```

```
    #define EndFor    }

       Set ^test(1)=111
       Set ^test(2)=222
       Set ^test(3)=333

       $$$EachFor(key,^test)
           Write "key: ",key,!
           Write "value: ",^test(key),!
       $$$EndFor
}
```

Así es como se ve en el código INT:

```
zTest3() public {
       Set ^test(1)=111
       Set ^test(2)=222
       Set ^test(3)=333
       Set %mmmu1=$name(^test)
       Set key=""
       For {
           Set key=$o(@%mmmu1@(key))
           Quit:key=""
           Write "key: ",key,!
           Write "value: ",^test(key),!
       } }
```

¿ Qué está pasando en estas macros?

1. Escribe el nombre del global a una nueva variable %mmmu1 (función $name)
2. La clave asume el valor inicial de cadena vacía
3. Comienza el ciclo de iteración
4. El siguiente valor de la clave se asigna usando las funciones indirection y $order
5. Post-condition se utiliza para verificar si la clave ha asumido un valor "" ; si es así, la iteración se completa y el ciclo termina
6. Se ejecuta código de usuario arbitrario, en este caso, salida de clave y valor
7. Se cierra el ciclo

El terminal muestra lo siguiente cuando se llama a este método:

```
key: 1
value: 111
key: 2
value: 222
key: 3
value: 333
```

Si está utilizando listas y matrices heredadas de la clase %Collection.AbstractIterator, puede escribir un iterador similar para ello.

Ejemplo 4

Otra capacidad de las macros es la ejecución de código arbitrario de ObjectScript en la etapa de compilación y la sustitución de sus resultados en lugar de una macro. Creemos una macro para mostrar el tiempo de compilación:

```
ClassMethod Test4()
{
      #Define CompTS ##Expression("""Compiled: " _ $ZDATETIME($HOROLOG) _ """,!")
      Write $$$CompTS
}
```

Que se transforma en el siguiente código INT:

```
zTest4() public {
      Write "Compiled: 18.10.2016 15:28:45",! }
```

El terminal mostrará la siguiente línea cuando se llame a este método:

```
Compiled: 18.10.2015 15:28:45
```

##Expression ejecuta el código y sustituye el resultado. Los siguientes elementos del lenguaje ObjectScript se pueden utilizar para la entrada:

- Strings: "abc"
- Routines: $$Label^Routine
- Class methods: ##class(App.Test).GetString()
- Funciones COS: $name(var)
- Y combinaciones de estos elementos

Ejemplo 5

Directivas del pre procesador #If, #ElseIf, #Else, #EndIf se utilizan para seleccionar el código fuente durante la compilación, según el valor de la expresión que sigue una directiva. Por ejemplo, este método:

```
ClassMethod Test5()
{
    #If $SYSTEM.Version.GetNumber()="2016.2.0" && $SYSTEM.Version.GetBuildNumber()="7
36"
        Write "You are using the latest released version of Caché"
    #ElseIf $SYSTEM.Version.GetNumber()="2017.1.0"
        Write "You are using the latest beta version of Caché"
    #Else
        Write "Please consider an upgrade"
    #EndIf
}
```

Se compilará en el siguiente código INT en Caché versión 2016.2.0.736:

```
zTest5() public {
   Write "You are using the latest released version of Caché"
}
```

Y lo siguiente se mostrará en la terminal:

```
You are using the latest released version of Caché
```

Si usamos Caché descargado del portal beta, el código INT compilado tendrá un aspecto diferente:

```
zTest5() public {
    Write "You are using the latest beta version of Caché"
}
```

Lo siguiente se mostrará en la terminal:

```
You are using the latest beta version of Caché
```

Las versiones anteriores de Caché compilarán el siguiente código INT con una sugerencia para actualizar el programa:

```
zTest5() public {
    Write "Please consider an upgrade"
}
```

El terminal mostrará el siguiente texto:

```
Please consider an upgrade
```

Esta capacidad puede ser útil, por ejemplo, en situaciones en las que desea garantizar la compatibilidad de la aplicación cliente con versiones anteriores y nuevas, donde se pueden utilizar las nuevas características de Caché. Directivas del pre procesador #IfDef, #IfNDef tienen el mismo propósito al verificar la existencia o ausencia de una macro, respectivamente.

Conclusiones

Las macros pueden hacer que su código sea más legible al simplificar las construcciones de uso frecuente y ayudarlo a implementar parte de la lógica de negocios de su aplicación en la etapa de compilación, reduciendo así la carga en tiempo de ejecución.

Enlaces

- Acerca de la compilación
- Lista de directivas de preprocesador
- Lista de macros del sistema
- Clase con ejemplos de este artículo.

#Compilador #Consejos y trucos #ObjectScript #Principiante #Terminal #Caché

URL de fuente: https://es.community.intersystems.com/post/macros-en-intersystems-cach%C3%A9