Artículo
Ricardo Paiva · 9 feb, 2023 Lectura de 4 min

Open Exchange

iris-tripleslash - ¡Toquemos juntos!

Hola a todos,

Aquí estamos de nuevo. Nuevo año, nuevo concurso, nuevo proyecto, viejos motivos.

¡Triple Slash ya está en casa!

1999, el año que aprendí a programar, mi primer "if," mi primer "Hello world."

Aún recuerdo a mi profesor explicándonos en aquella clase el sencillo "while" y cómo podemos saber si se cumplió una condición específica. ¿Te acuerdas, @Renato Banzai? El profesor Barbosa, un tipo único.

Desde entonces, me ha encantado la idea de programar, transformando ideas en proyectos, en algo útil. Pero todos sabemos que para crear algo, necesitamos asegurarnos de que está funcionando; necesitamos no solo crear, sino también probar si funciona y si no se rompe si añadimos algo nuevo.

Y para ser honesto con todos vosotros, hacer pruebas es aburrido. Al menos para mí, no tengo nada contra vosotros si os gusta.

Usando una analogía, podría decir que crear métodos de prueba es como limpiar la casa o planchar la ropa. Es aburrido, pero necesario.

Con estas ideas en mente, ¿por qué no desarrollar una forma mejor y más sencilla de probar?

Así que, inspirado por el estilo de elixir y por esta idea de InterSystems Ideas (gracias, @Evgeny Shvarov), intentamos mejorar el proceso de prueba y convertirlo en una tarea divertida otra vez.

Simplificamos el %UnitTest y para mostraros cómo usar TripleSlash para crear pruebas unitarias (*unit tests*), vamos a utilizar un ejemplo sencillo.

Pongamos que tienes la siguiente clase y método del que te gustaría escribir una prueba unitaria:

```
Class dc.sample.ObjectScript
{
ClassMethod TheAnswerForEverything() As %Integer
{

   Set a = 42
   Write "Hello World!",!

   Write "This is InterSystems IRIS with version ",$zv,!

   Write "Current time is: "_$zdt($h,2)
```

```
   Return a

}
}
```

Como se puede ver, el método **TheAnswerForEverything()** solo devuelve el número **42**. Así que vamos a marcar en la documentación del método cómo TripleSlash debería crear una prueba unitaria para este método:

```
/// A simple method for testing purpose.
///
/// <example>
/// Write ##class(dc.sample.ObjectScript).Test()
/// 42
/// </example>
ClassMethod TheAnswerForEverything() As %Integer
{
    ...
}
```

Las pruebas unitarias deben estar todas en una etiqueta ***<example></example>***. Se puede añadir cualquier tipo de documentación, pero todas las pruebas tienen que estar dentro de ese tipo de etiqueta.

Ahora, arranca una instancia de IRIS e inicia una sesión de terminal, ve al *namespace* **IRISAPP**, crea una instancia de la clase **Core** pasando el nombre de clase (o su nombre de paquete para todas sus clases) y después ejecuta el método **Execute()**:

```
USER>ZN "IRISAPP"
IRISAPP>Do ##class(iris.tripleSlash.Core).%New("dc.sample.ObjectScript").Execute()
```

TripleSlash interpretará esto como "Dado el resultado del método **Test()**, afirmo que es igual a **42**". Así que se creará una nueva clase dentro de la prueba unitaria:

```
Class iris.tripleSlash.tst.ObjectScript Extends %UnitTest.TestCase
{

Method TestTheAnswerForEverything()
{
    Do $$$AssertEquals(##class(dc.sample.ObjectScript).TheAnswerForEverything(), 42)
}

}
```

Ahora, vamos a añadir un nuevo método para probar otras formas de decir a TripleSlash cómo escribir las pruebas unitarias.

```
Class dc.sample.ObjectScript
{

ClassMethod GuessTheNumber(pNumber As %Integer) As %Status
{
    Set st = $$$OK
    Set theAnswerForEveryThing = 42
```

```
    Try {
        Throw
:(pNumber '= theAnswerForEveryThing) ##class(
%Exception.StatusException).%New("Sorry, wrong number...")
    } Catch(e) {
        Set st = e.AsStatus()
    }

    Return st

}

}
```

Como se puede ver, el método **GuessTheNumber()** espera un número, devuelve **$$$OK** solo cuando se pasa como argumento el número **42** o un error para cualquier otro valor. Así que vamos a marcar en la documentación del método cómo TripleSlash debería crear una prueba unitaria para este método:

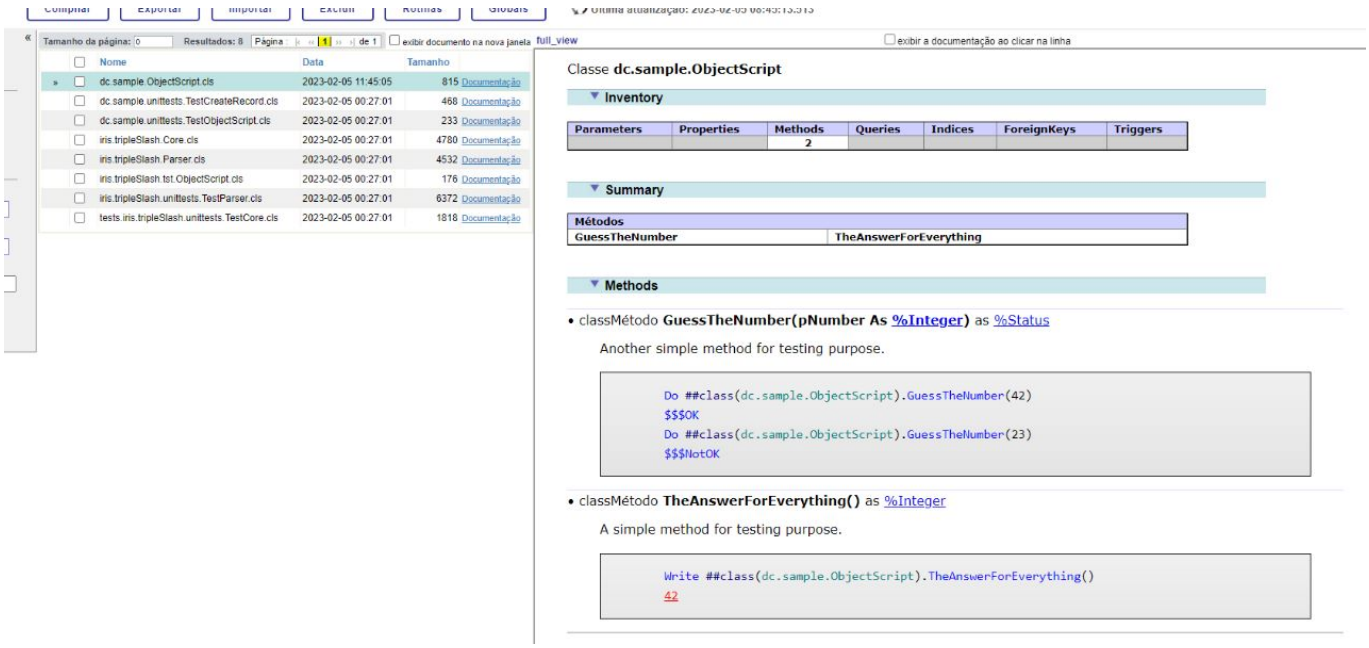
```
/// Another simple method for testing purpose.
///
/// <example>
/// Do ##class(dc.sample.ObjectScript).GuessTheNumber(42)
/// $$$OK
/// Do ##class(dc.sample.ObjectScript).GuessTheNumber(23)
/// $$$NotOK
/// </example>
ClassMethod GuessTheNumber(pNumber As %Integer) As %Status
{
    ...
}
```

Ejecuta otra vez el método **Execute()** y verás un nuevo método de prueba en la clase de prueba unitaria **iris.tripleSlash.tst.ObjectScript**:

```
Class iris.tripleSlash.tst.ObjectScript Extends %UnitTest.TestCase
{

Method TestGuessTheNumber()
{

    Do $$$AssertStatusOK(##class(dc.sample.ObjectScript).GuessTheNumber(42))
    Do $$$AssertStatusNotOK(##class(dc.sample.ObjectScript).GuessTheNumber(23))
}

}
```

En este momento, las siguientes afirmaciones están disponibles: **$$$AssertStatusOK**, **$$$AssertStatusNotOK** y **$$$AssertEquals**.

TripleSlash nos permite generar pruebas desde ejemplos de código que se encuentran en las descripciones de métodos. Te ayuda a *matar dos pájaros de un tiro*, mejorando tu documentación de clase y creando automatización de pruebas.



Reconocimiento

Una vez más, me gustaría agradecer todo el apoyo de la Comunidad en todas las aplicaciones que creamos!

#Concurso #Framework #Prueba #Portal de ideas de InterSystems #InterSystems IRIS #InterSystems IRIS for Health #Open Exchange
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/iris-tripleslash-%C2%A1toquemos-juntos