Artigo
Henrique Dias · Fev. 9, 2023 4min de leitura

Open Exchange

iris-tripleslash - vamos detonar geral

Fala galera, beleza?

iiii lá vamos nós. Ano novo, nova competição, novo projeto, velhos motivos.

Triple Slash na área!

Aprendi o meu primeiro if, hello world, em 1999.

Ainda lembro do meu professor tentando explicar para turma toda como um simples "while" funcionava para descobrir se uma determinada condição havia sido atingida. @Renato Banzai ainda lembra do professor Barbosa tentando chegar a porta na explicação "passo-a-passo"?

De lá pra cá, sempre amei a ideia de programar, criar coisas, transformar ideias em projetos, em coisas úteis. Mas para criar algo, você precisa ter certeza de que isso funciona. Não apenas sair fazendo coisas novas, mas garantir que o que você criou funciona, mesmo adicionando novas funcionalidades.

E sendo super transparente, eu acho a parte de testes chata demais! Imagina uma parada chata, é o que teste é pra mim. Se você curte essas paradas, nada contra, mas não é minha praia.

Comparando mal e porcamente, seria como limpar a casa. Todo mundo curte um ambiente limpinho, mas limpar a casa, passar roupa, etc é uma baita chatice.

Com isso em mente, pensamos em facilitar a vida de todo desenvolvedor na hora de testar suas aplicações.

Inspirando no estilo do elixir e nessa ideia da InterSystems Ideas (Valeu @Evgeny Shvarov)! Nós tentamos melhorar o processo de testes e torná-lo mais fácil e agradável de se fazer.

Simplificamos o %UnitTest e para mostrar como utilizar o TripleSlash para criar seus testes unitários, se liga no exemplo abaixo:

Vamos supor que você tenha a seguinte classe e método e queira escrever um teste unitário:

```
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 podemos ver o método **TheAnswerForEverything()** apenas retorna o número **42**. Então, vamos documentar o método e como o TripleSlash deve criar o teste unitário:

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

Testes unitários devem estar entre a tag ***<example></example>***. Você pode colocar todo o tipo de documentação, mas todos os testes devem estar obrigatoriamente dentro da tag.

Agora, vamos iniciar uma sessão no IRIS, ir para o Namespace **IRISAPP**, criar uma instância da classe **Core** passando o nome da classe (ou o nome do pacote de classes para todas elas) e executar o método **Execute()** :

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

TripleSlash vai interpretar mais ou menos como "Dado o resultado do método **Test()**, afirmamos que é igual a **42**". Então, a nova classe será criada:

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

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

}
```

Agora vamos adicionar um novo método para testar outras coisas e falar para o TripleSlash como escrever esses testes.

```
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 podem ver, o método **GuessTheNumber()** espera um número, retorna um **$$$OK** apenas quando o número **42** é passado e devolve um erro para qualquer outro valor. Então vamos falar como queremos que o TripleSlash crie o teste unitário:

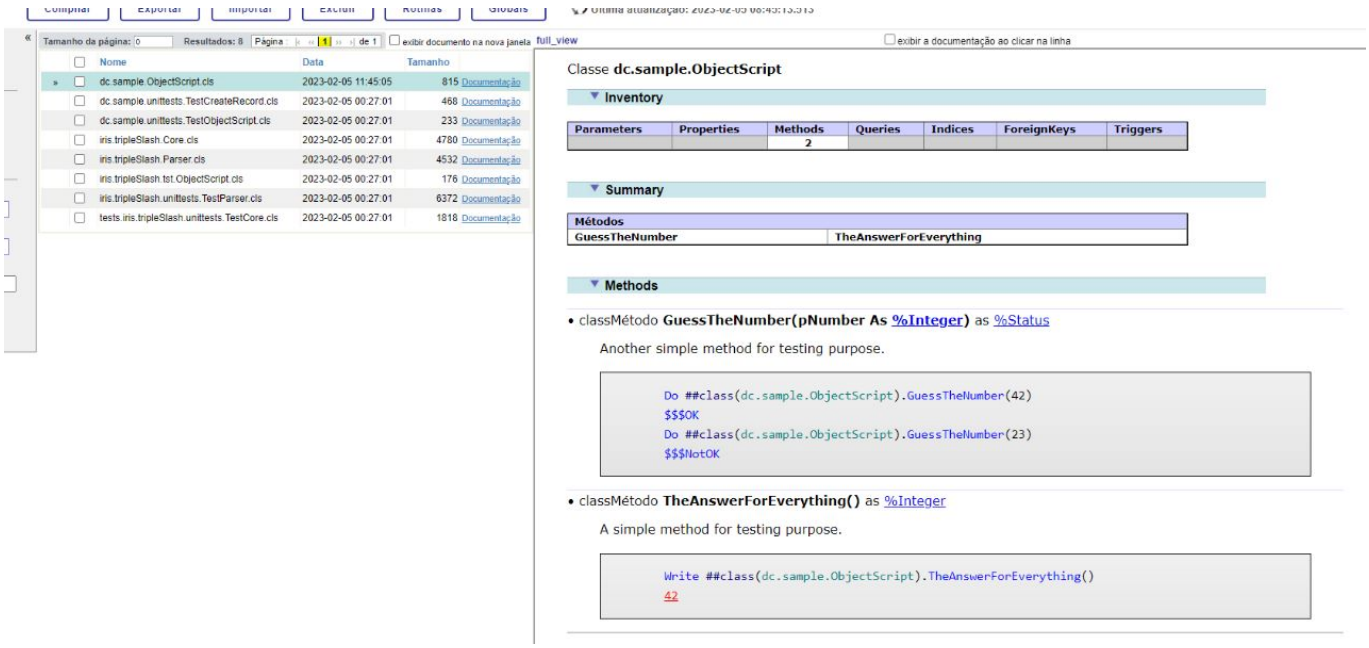
```
/// Another simple method for testing purpose.
///
/// <example>
/// Do ##class(dc.sample.ObjectScript).GuessTheNumber(42)
/// $$$OK
/// Do ##class(dc.sample.ObjectScript).GuessTheNumber(23)
/// $$$NotOK
/// </example>
ClassMethod GuessTheNumber(pNumber As %Integer) As %Status
{
    ...
}
```

Executando novamente o método **Execute()** e verá um novo método de teste na sua classe de teste **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))
}

}
```

Atualmente temos suporte para: **$$$AssertStatusOK**, **$$$AssertStatusNotOK** and **$$$AssertEquals**.

TripleSlash nos permite gerar testes a partir de exemplos de códigos encontrados na documentação dos métodos. Isso ajuda a matar 2 coelhos com uma cajadada só, melhorando sua documentação de classe e criando um teste unitário.



Agradecimentos

Uma vez mais, queremos agradecer a todos que vem nos dando apoio na comunidade com as aplicações que criamos.

Se você achou nosso aplicativo interessante, pensa com carinho para votar em iris-tripleslash e nos ajude a continuar contribuindo para o crescimento da comunidade a e aseguir nesse caminho!

#Concurso #Framework #Testes #Portal de ideias da InterSystems #InterSystems IRIS #InterSystems IRIS for Health #Open Exchange
Confira o aplicativo relacionado no InterSystems Open Exchange

URL de origem:https://pt.community.intersystems.com/post/iris-tripleslash-vamos-detonar-geral