文章
姚 鑫 · 五月 16, 2021 阅读大约需 分钟

第二章 使用%UnitTest进行单元测试

第二章 使用%UnitTest进行单元测试

本教程的第二部分介绍了如何使用%UnitTest包对InterSystems IRIS代码进行单元测试。完成教程的这一部分后，将能够：

- 解释%UnitTest包中三个主要类的角色。
- 列出基%UnitTest包的单元测试类和方法的要求。
- 创建并执行方法的单元测试。
- 浏览%UnitTest.Manager创建的测试报告。
- 执行单元测试时，使用%UnitTest.TestCase方法初始化和还原数据库数据。

什么是%UnitTest？

%UnitTest包是一组为IRIS提供测试框架的类。在结构上，它类似于xUnit测试框架。%UnitTest为创建和执行以下项的单元测试提供类和工具：

- 类和方法
- ObjectScript例程(routines)
- InterSystems SQL脚本
- Productions

创建和执行单元测试套件

以下是创建和执行一套单元测试的基本步骤：

1. 创建一个(或多个)包含要测试的方法的类。
2. 创建扩展%UnitTest.TestCase的测试类(或多个测试类)。
3. 将方法添加到将测试方法输出的测试类。在每个方法中至少使用一个断言(AssertX宏)。每个测试方法名称都以Test开头。
4. 将测试类导出到文件。
5. 打开终端并切换到包含要测试的类的名称空间。为^UnitTestRoot分配一个字符串，该字符串包含包含导出的测试类文件的目录的父目录的路径。
6. 在终端中，运行%UnitTest.Manager.RunTest，向其传递包含测试类文件的(子)目录的名称。
7. 查看测试报告。终端中的输出包括网页的URL，该网页以易于阅读的表格形式显示结果。

%UnitTest类

此表描述了用于为InterSystems IRIS类和方法创建和执行单元测试的主要%UnitTest类。

- TestCase 扩展此类以创建包含测试方法的类。如果一个或多个AssertX方法返回False，则测试失败；否则测试通过。将使用关联的宏调用AssertX方法。这些方法和宏是：

    - AssertEqualsViaMacro-如果表达式相等，则返回TRUE。使用$$$AssertEquals宏调用。
    - AssertNotEqualsViaMacro-如果表达式不相等，则返回TRUE。使用$$$AssertNotEquals宏调用。
    - AssertStatusOKViaMacro-
      如果返回的状态代码为1，则返回TRUE。使用$$$AssertStatusOK宏调用。
    - AssertStatusNotOKViaMacro-
      如果返回的状态码为0，则返回TRUE。使用$$$AssertStatusNotOK宏调用。
    - AssertTrueViaMacro-如果表达式为TRUE，则返回TRUE。使用$$$AssertTrue宏调用。
    - AssertNotTrueViaMacro-如果表达式不为TRUE，则返回TRUE。使用$$$AssertNotTrue宏调用。
    - AssertFilesSameViaMacro-如果两个文件相同，则返回TRUE。使用$$$AssertFilesSame宏调用。
    - LogMessage-将日志消息写入^UnitTestLog全局。使用$$$LogMessage宏调用。
    - 设置和拆除条件的方法包括：
    - OnBeforeOneTest-紧接在测试类中的每个测试方法之前执行。
    - OnBeforeAllTests-在测试类中的任何测试方法之前执行一次。
    - OnAfterOneTest-在测试类中的每个测试方法之后立即执行。
    - OnAfterAllTests-在测试类中的所有测试方法执行完毕后执行一次。
- Manager 使用此类启动测试。其方法包括：
    - RunTest -在目录中执行一个测试或一组测试。
    - DebugRunTestCase-执行一个测试或一组测试，而不加载或删除任何测试类。
- Report 定义报告执行一个测试或一组测试的结果的网页。

断言方法和宏

单元测试的主要测试操作来自AssertX方法及其关联宏。将直接调用宏来测试方法的输出。宏测试方法是否为给定的输入创建所需输出。只要AssertX宏返回FALSE(或以错误结束)，包含它的测试就会失败。

在创建代码时，请计划将创建的单元测试以测试代码。在这里的示例中，已经创建了一个名为TestMe的类，其中包含一个名为Add的方法。现在想测试一下新的TestMe类，看它是否工作。

以下命令运行AssertEquals宏以测试Add方法的输入(2，2)是否等于4。

```
Do $$$AssertEquals(##class(MyPackage.TestMe).Add(2,2),4, "Test Add(2,2)=4")
```

AssertEquals宏比较两个值并接受三个参数：

1. ##class(MyPackage.TestMe).Add(2，2)-第一个值是以2，2作为输入进行测试的方法。
2. 4-第二个值。
3. "Test Add(2,2)=4"-写在结果页上的文本说明。(此参数不影响测试。如果不包含测试描述，该类将使用求值的表达式创建一个测试描述。)

以下是用于测试对象是否正确保存AssertStatusOK宏的示例。

```
Do $$$AssertStatusOK(contact.%Save(),"Saving a Contact")
```

此AssertStatusOk宏计算方法返回的状态。如果为1，则测试通过。

1. Contact.%Save-返回状态代码的表达式。
2. "Saving a Contact" -文本说明。这是测试报告的文档。这不会影响测试。

创建要在示例中使用的类

要完成动手示例，请使用Atelier创建以下类：MyPackage.TestMe和MyPackage.Contact。

- MyPackage.TestMe

```
Class MyPackage.TestMe Extends %RegisteredObject
{

ClassMethod Add(arg1 As %Integer, arg2 As %Integer) As %Integer
{

    Return arg1 + arg2
}

ClassMethod CreateContact(name As %String, type As %String) As MyPackage.Contact
{

    Set contact = ##class(MyPackage.Contact).%New()
    Set contact.Name=name
    Set contact.ContactType=type
    Return contact
}

ClassMethod GetContactsByType(type As %String) As %ListOfObjects
{

    Set list=##class(%Library.ResultSet).%New()
}

}
```

- MyPackage.Contact

```
Class MyPackage.Contact Extends (%Persistent, %Populate, %XML.Adaptor)
{

/// ????????: Personal or Business
Property ContactType As %String(TRUNCATE = 1, VALUELIST = ",Business,Personal");

/// ????????
Property Name As %String(POPSPEC = "Name()", TRUNCATE = 1) [ Required ];

Query ByContactType(type As %String) As %SQLQuery(CONTAINID = 1)
{
    SELECT %ID FROM Contact
    WHERE (ContactType = :type)
    ORDER BY Name
}

Storage Default
{
<Data name="ContactDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>ContactType</Value>
</Value>
<Value name="3">
<Value>Name</Value>
```

```
</Value>
</Data>
<DataLocation>^MyPackage.ContactD</DataLocation>
<DefaultData>ContactDefaultData</DefaultData>
<IdLocation>^MyPackage.ContactD</IdLocation>
<IndexLocation>^MyPackage.ContactI</IndexLocation>
<StreamLocation>^MyPackage.ContactS</StreamLocation>
<Type>%Storage.Persistent</Type>
}

}
```

示例：创建并导出测试类

类MyPackage.TestMe包含一个名为Add的方法，该方法将两个整数相加。在此示例中，将创建并运行单元测试以检查Add方法是否正确地将两个整数相加。

创建将包含单元测试的测试类。以下是方法：

1. 使用Atelier在MyPackage包中创建名为Tests的新类。测试必须扩展%UnitTest.TestCase。
2. 添加以下名为TestAdd并编译测试的方法：

```
Class MyPackage.Tests Extends %UnitTest.TestCase
{

Method TestAdd()
{
 do $$$AssertEquals(##class(MyPackage.TestMe).Add(2,2),4, "Test Add(2,2)=4")
 do $$$AssertNotEquals(##class(MyPackage.TestMe).Add(2,2),5,"Test Add(2,2)'=5")
}

}
```

3. 将类测试导出到单元测试目录中的XML文件。如果尚未创建测试目录，请创建一个。此示例使用 `C: \ unittests \ mytests \ 。

a. 在Atelier中，单击文件>导出。

b. 在“Atelier ”下，单击“旧版XML文件”。单击下一步

c. 选择项目Test.cls和c: \ unittests \ mytests \ 目录。

d. 单击Finish(完成

e. Atelier将测试类导出到C: \ unittests \ mytests \ cls \ MyPackage。

注 意，目录名(在本例中为mytest)是一套测试的名称，也是^UnitTestRoot指定的目录的子级。运行Manager.RunTest(“mytest”)运行存储在mytest目录中的所有测试。

注 意：还可以将测试类导出为.cls文件，而不是XML文件。也可以简单地从Atelier工作区复制它们，而不是导出它们。

源码

#SQL #Caché #InterSystems IRIS #InterSystems IRIS for Health

源 URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E4%BA%8C%E7%AB%A0-%E4%BD%BF%E7%94%A8unittest%E8%BF%9B%E8%A1%8C%E5%8D%95%E5%85%83%E6%B5%8B%E8%AF%95