InterSystems IRIS の %UnitTest フレームワーク

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

InterSystems IRIS の %UnitTest フレームワーク
InterSystems Version 2024.1 2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

目次

テーブル一覧

1

InterSystems IRIS の %UnitTest フレームワークについて

%UnitTest は、InterSystems IRIS のユニット・テスト・フレームワークです。xUnit フレームワークに精通している開発者は、%UnitTest 内に含まれる構造には馴染みがあることでしょう。

- %UnitTest.TestCase クラスを拡張し、テスト・メソッドを追加して、ユニット・テストを作成します。詳細は、"%UnitTest.TestCase クラスの拡張" を参照してください。
- %UnitTest.TestCase クラスの特殊なクリーンアップ・メソッドと準備メソッドにコードを追加して、準備タスクとクリーンアップ・タスクを実行します。

  詳細は、"%UnitTest.TestCase クラスの準備メソッドとクリーンアップ・メソッド" を参照してください。
- %UnitTest.Manager クラスの RunTests() メソッドを使用して、テストを実行します。一般的な結果がターミナル・ウィンドウに表示されます。詳細は、"%UnitTest.Manager メソッドを使用したユニット・テストの実行" を参照してください。
- 管理ポータルのテスト結果 Web ページで詳細を確認します。詳細は、"管理ポータルでの %UnitTest レポートの表示" を参照してください。

%UnitTest パッケージには、以下のクラスが含まれます。

- TestCase ― このクラスを拡張してテスト・クラスを作成してから、ユニット・テストを含むクラス・メソッドを追加します。
- Manager ― ユニット・テストを実行するメソッドが含まれます。
- Report ― テスト結果の Web ページを含め、テストの出力を制御します。

2

テスト・ケースの作成 : %UnitTest.TestCase クラス

これは、%UnitTest フレームワークを使用してユニット・テストを設定する一般的なワークフローです。

1. %UnitTest.TestCase クラスを拡張し、テスト対象のメソッドごとに 1 つのテスト・メソッドを追加します。テスト・メソッド名は、Test という単語で始まる必要があります。"%UnitTest.TestCase クラスの拡張" を参照してください。
2. 1 つのテスト・メソッドに複数のテストを含めることができます。通常、テスト・メソッドには、テスト対象のメソッドの各側面につき 1 つのテストが含まれます。各テスト・メソッド内で、$$$AssertX マクロを使用して 1 つ以上のテストを作成します。通常、マクロはテスト対象のメソッドを呼び出し、その出力を期待値と比較します。期待値がマクロの出力と一致する場合、テストは成功と見なされます。"%UnitTest.TestCase クラスのマクロ" を参照してください。
3. 準備メソッドとクリーンアップ・メソッドにコードを追加して、必要なタスクを実行します。例えば、テストでリストから要素を削除する場合、まずそのリストが存在し、そこに削除する要素が含まれている必要があります。"%UnitTest.TestCase クラスの準備メソッドとクリーンアップ・メソッド" を参照してください。

   注釈　準備メソッドとクリーンアップ・メソッドはしばしば、セットアップ・メソッドおよびティアダウン・メソッドとも呼ばれます。

2.1 %UnitTest.TestCase クラスの拡張

%UnitTest.TestCase を拡張するクラスを作成し、ユニット・テストを実行するテスト・メソッドを含めます。このプロセスは、特定のテスト・ニーズに対応できるよう柔軟な設計になっています。

ほとんどの場合、テスト・メソッドを追加し、プロパティも追加します。テスト・メソッドは、'Test' で始まる名前のメソッドを探して実行する %UnitTest.Manager クラスの RunTests() メソッドによって実行されます。クラスに他のヘルパー・メソッドを追加することはできますが、RunTests() を呼び出したときにメソッドがユニット・テストとして実行されるのは、メソッド名が 'Test' で始まる場合のみです。

注釈　テスト・メソッドはアルファベット順に実行されます。例えば TestAssess() は TestCreate() より前に実行されます。

テスト・メソッド内では、1 つ以上のテストを作成します。各テストに $$$AssertX マクロを使用します。$$$AssertX マクロの詳細は、"%UnitTest.TestCase クラスのマクロ" を参照してください。

テストするクラス・メソッドごとにテスト・メソッドを作成することもできます。例えば、MyPackage.MyClassToBeTested クラスに複数のテストを呼び出す Add() メソッドが含まれる場合、必要なテストを実行するコードを含む MyTests.TestAdd() テスト・メソッドを作成できます。

オブジェクト・インスタンスをテストすることもできます。この場合、オブジェクトのプロパティと機能が正しいことを確認するテストを含めることができる、MyTests.TestMyObject() のようなメソッドを作成します。

テスト・メソッドを作成するほかに、拡張クラスではプロパティを作成することもできます。これにより、テスト・メソッドは情報を共有できます。プロパティを追加する際は、以下の点を考慮してください。

- クラス自体でカスタム・プロパティを宣言します。
- `..<property>` 構文を使用し、準備メソッド OnBeforeOneTest() および OnBeforeAllTests() にコードを追加してプロパティを設定します。
- `..<property>` 構文を使用し、テスト・メソッドやクリーンアップ・メソッド OnAfterOneTest() および OnAfterAllTests() にコードを追加してプロパティにアクセスします。

  注釈　例えば、カスタム・プロパティが PropertyValue と呼ばれる場合、`..PropertyValue` を使用して設定またはアクセスできます。

2.1.1 例 : 拡張 %UnitTest.TestCase クラス

```
Class MyPackage.MyClassToBeTested
{
  Method Add (Addend1 as %Integer, Addend2 as %Integer) As %Integer
  {
    Set Sum = Addend1 + Addend2
    Return Sum
  }
}

Class MyPackage.MyTests Extends %UnitTest.TestCase
{
  Method TestAdd()
  {
    do $$$AssertEquals(##class(MyPackage.MyClassToBeTested).Add(2,3),5, "Test 2+3=5")
    do $$$AssertNotEquals(##class(MyPackage.MyClassToBeTested).Add(3,4),5, "Test 3+4 '= 5")
  }
}
```

2.2 %UnitTest.TestCase クラスのマクロ

各テスト・メソッド内で、以下のいずれかの `$$$AssertX` マクロを使用して、クラス・メソッドのテスト可能な側面をテストします。例えば、テスト・メソッドが Add() メソッドをテストするように設計されている場合、これには、`$$$AssertEquals` を使用して 2+3 が 5 になることを確認するテストと、`$$$AssertNotEquals` を使用して 3+4 が 5 にならないことを確認するテストを含めることができます。

必要なテスト結果に最も一致するマクロを選択します。この原則を考慮する方法として、アサーションが成功するという観点からテストを作成することもできます。2 つの値が等しくなることを期待する場合は `$$$AssertEquals` を使用し、等しくならないことを期待する場合は `$$$AssertNotEquals` を使用します。

指定した `$$$AssertX` マクロが False を返す場合、テストは不合格で、そうでない場合、テストは合格です。

`$$$AssertX` マクロは、次の引数を取ることができます。

- arg1 ー 通常、テスト対象のメソッドの出力、またはその出力から計算された値です。
- arg2 ー 存在する場合、マクロが arg1 と比較する値です。

- test_description ー 表示されたテスト結果のリストにある文字列で、マクロが何をテストしたかを示します。これは、テストの結果には影響しません。この引数には、連結、変数、メソッドを含めることができます。例えば、値は以下のようになります。

  "Failed to create" _ maxObjects _ "objects: " _ $system.Status.GetErrorText(status)

$$$AssertEquals (arg1, arg2, test_description)

arg1 と arg2 が等しい場合、True を返します。

```
do $$$AssertEquals (##class(MyPackage.MyClassToBeTested).Add(2,3), 5, "Test Add(2,3) = 5")
```

$$$AssertNotEquals (arg1, arg2, test_description)

arg1 と arg2 が等しくない場合、True を返します。

```
do $$$AssertNotEquals (##class(MyPackage.MyClassToBeTested).Add(3,4), 5, "Test Add(3,4) '= 5")
```

$$$AssertStatusOK (arg1, test_description)

返されたステータス・コードが 1 の場合、True を返します。

```
do $$$AsserStatusOK(##class(MyPackage.MyClassToBeTested).SaveContact(valid_contact_ID),
        "Test that valid contact is saved")
```

$$$AssertStatusNotOK (arg1, test_description)

返されたステータス・コードが 1 でない場合、True を返します。

```
do $$$AssertStatusNotOK(##class(MyPackage.MyClassToBeTested).SaveContact(invalid_contact_ID),
        "Test that invalid contact is not saved")
```

$$$AssertTrue (arg1, test_description)

式が True の場合、True を返します。

```
do $$$AssertStatusTrue(##class(MyPackage.MyClassToBeTested).IsContactValid(valid_contact_ID),
        "Test that valid contact is valid")
```

$$$AssertNotTrue (arg1, test_description)

式が True でない場合、True を返します。

```
do $$$AssertStatusNotTrue(##class(MyPackage.MyClassToBeTested).IsContactValid(invalid_contact_ID),
        "Test that invalid contact is not valid")
```

$$$AssertFilesSame (arg1, arg2, test_description)

2 つのファイルが同じである場合、True を返します。

```
do $$$AssertFilesSame(##class(MyPackage.MyClassToBeTested).FetchFile(URL), control_file,
        "Test that fetched file is identical to control file")
```

$$$AssertFilesSQLUnorderedSame (arg1, arg2, test_description)

SQL クエリ結果を含む 2 つのファイルに、順序付けられていない同じ結果が含まれる場合、True を返します。

```
do $$$AssertFilesSQLUnorderedSame(output.log,reference.log,"Comparing output.log to reference.log")
```

$$$AssertSuccess(test_description)

無条件で成功をログに記録します。このアサーションは、$$$AssertTrue に 1 を渡す規則を置き換えることを目的としています。

$$$AssertFailure(test_description)

無条件で失敗をログに記録します。このアサーションは、$$$AssertTrue に 0 を渡す規則を置き換えることを目的としています。

$$$AssertSkipped(test_description)

test_description で説明されている理由でテストがスキップされたというメッセージをログに記録します。これは、例えばテストの前提条件が満たされていない場合に使用されます。

注釈　OnBeforeAllTests() は、このマクロをサポートしていません。OnBeforeAllTests() で $$$AssertSkipped を呼び出すと、誤検出が発生する可能性があります。

$$$LogMessage (message)

特定のテストとは無関係に、message の値をログ・エントリとして書き込みます。これは、例えばログにコンテキストや編成を提供する場合に非常に便利です。

```
do $$$LogMessage("-- ALL TEST OBJECTS CREATED --")
```

注釈　マクロの最新のリストは、クラス・リファレンスの "%UnitTest.TestCase" を参照してください。

2.3 %UnitTest.TestCase クラスの準備メソッドとクリーンアップ・メソッド

%UnitTest.TestCase には、テスト用の準備メソッドとクリーンアップ・メソッドが含まれます。これらのメソッドにコードを追加して、データベース接続の作成やテスト・データでのデータベースの初期化などの準備タスクを実行したり、データベース接続の終了やデータベースの状態のリストアなどのクリーンアップ・タスクを実行できます。

OnBeforeOneTest()

テスト・クラスの各テスト・メソッドの直前に実行します。

OnBeforeAllTests()

テスト・クラスのすべてのテスト・メソッドの前に 1 回だけ実行します。

OnAfterOneTest()

テスト・クラスの各テスト・メソッドの直後に実行します。

OnAfterAllTests()

テスト・クラスのすべてのテスト・メソッドが実行された後に 1 回だけ実行します。

2.3.1 例 : 準備メソッド

このメソッドのコードは、テスト・スイートの実行前に 1 回実行されます。テスト中に使用する単一コンタクトを作成します。準備タスクを複数回実行するには (スイートの各テストの前に 1 回ずつ)、代わりに OnBeforeOneTest() にコードを追加します。

```
Method OnBeforeAllTests()
{
  Do ##class(MyPackage.Contact).Populate(1)
  Return $$$OK
}
```

2.3.2 例 : クリーンアップ・メソッド

このメソッドのコードは、テスト・スイート全体の実行後に 1 回実行されます。テストの完了後、エクステントのすべてのコンタクトを削除します。クリーンアップ・タスクを複数回実行するには (スイートの各テストの後に 1 回ずつ)、代わりに OnAfterOneTest() にコードを追加します。

```
Method OnAfterAllTests()
{
  Do ##class(MyPackage.Contact).%KillExtent()
  Return $$$OK
}
```

3

%UnitTest.Manager メソッドを使用したユニット・テストの実行

%UnitTest.Manager クラスに含まれるメソッドを使用してテストを開始します。

これは、%UnitTest フレームワークを使用してユニット・テストを実行する一般的なワークフローです。

1. ^UnitTestRoot グローバルを設定して、テストの場所をシステムに知らせます。

   ```
   USER>set ^UnitTestRoot = "C:¥UnitTests"
   ```

2. %UnitTest.Manager クラスの RunTests() メソッドまたは DebugRunTestCase() メソッドを使用して、テストを実行します。

   ```
   USER>do ##class(%UnitTest.Manager).RunTests("MyTests")
   ```

3. テストの結果を表示します。

注釈　既定では、RunTests() は ^UnitTestRoot ディレクトリ内で見つかったすべてのテスト・クラスをロードしてコンパイルし、含まれているすべてのテストを実行して、メモリから削除します。

ただし、これがコードを開発する際の最も効率的なパラダイムではない場合がありますつまり、テスト対象のメソッドに小さな変更を加えるたびに、テストを再読み込みおよび再コンパイルしたくない場合もあります。そのため、ユニット・テスト・クラスは多くの場合、外部に格納されます。RunTests() メソッドの引数を使用して、テストをロードするかどうか、どこからロードするか、それらを削除するかどうか、およびその他の考慮事項を明示的に制御できます。詳細は、"RunTest() と DebugRunTestCase() の引数" を参照してください。

3.1 %UnitTest テスト実行メソッド

ユニット・テストを実行する際の既定の動作では、テストは InterSystems IRIS にロードされ、コンパイルされ、実行された後、削除されます。これにより、テスト・コードによって InterSystems IRIS ネームスペースが乱雑になるのを防ぎます。この既定の動作から外れるには、DebugRunTestCase() を使用するか、これらのいずれかのメソッドの qualifiers 引数にフラグを追加します。例えば、テスト・ケースをネームスペース内でローカルに開発し、変更を加えるたびに再読み込みしなくて済むようにすることができます。この場合は、qualifiers 引数の一部として `/nodelete` フラグを渡します。

RunTest (“testSpec”, “qualifiers”, “userparam”)

グローバル ^UnitTestRoot で指定されたディレクトリ内で、1 つのテストまたは一連のテストを実行します。テストが実行されたら、ロードされたすべてのテストとテスト・クラスを InterSystems IRIS から削除します。

```
USER>Do ##class(%UnitTest.Manager).RunTest("MyTests")
```

DebugRunTestCase (“testSpec”, “qualifiers”, “userparam”)

テスト・クラスをロードまたは削除せずに 1 つのテストまたは一連のテストを実行します。

```
USER>Do ##class(%UnitTest.Manager).DebugRunTestCase("MyTests","/display=none/debug","/log")
```

3.1.1 RunTest() と DebugRunTestCase() の引数

testSpec

テスト・スイートの 1 つ以上のテスト・クラスからテストを実行できます。testSpec 引数で、実行するテストおよびそれらの場所を指定します。値が渡されない場合、システムは ^UnitTestRoot とそのサブディレクトリ内のすべてのテスト・クラスを検索し、それらのクラス内のすべてのテストを実行します。一般的な形式は、`testSuite[:testCase[:testMethod][;testcase[:testmethod]]...]` です。testCase:testMethod のペアはセミコロンで区切られます。以下に例を示します。

```
MyTestSuite:MyFirstTestCase:MyFirstMethod;MySecondTestCase:MySecondMethod;MyThirdTestCase:MyThirdMethod
```

同じ構文を使用して、同じクラスから複数のテストを含めることができます。それには、各メソッドのクラスを指定します。以下に例を示します。

```
MyTestSuite:MyFirstTestCase:OneMethod;MyFirstTestCase:AnotherMethod
```

引数は以下のとおりです。

- testSuite

  テスト・クラス・ファイルを含むディレクトリ。このディレクトリは、^UnitTestRoot ディレクトリのサブディレクトリである必要があります。既定では、テスト・マネージャは、このディレクトリとそのサブディレクトリに含まれるすべてのファイルにあるすべてのテストを実行します。

- testCase

  含まれている各 testCase は、実行するテスト・メソッドを含む単一のクラスを指定します。構文は `PackageName.ClassName` です。この引数が指定されている場合、テスト・マネージャは、そのクラス内のテストのみを実行します。

- testMethod

  含まれている各 testMethod は、関連する testCase で示されたテスト・クラスの実行対象のメソッドを選択します。

qualifiers

テストを実行するためのさまざまなオプションを指定します。一般的な構文は、`/option1/option2/.../optionN` です。以下の修飾子を含めることができます。

- `/load`

  テストをディレクトリからロードします。`/noload` を使用すると、テストはロードされず、既に InterSystems IRIS に存在しているテストを実行します。既定値は `/load` です。

- `/run`

テストを実行します。/norun を使用すると、ロードは行われますが、テストは実行されません。既定値は /run です。

- /delete

  実行後にテスト・ケースを InterSystems IRIS から削除します。/nodelete を使用すると、クラスは保存されます。既定値は /delete です。

- /recursive

  指定されたディレクトリのサブディレクトリ内でテストを検索します。/norecursive を使用すると、サブディレクトリのテストは実行されません。既定値は /recursive です。

- /debug

  /debug を指定すると、最初のテストが失敗した後にテストは実行されません。ターミナルから実行すると、最初に失敗した後にターミナルはデバッグ・モードになります。この動作を避けるには、/nodebug を使用するか、単に /debug を使用しません。既定値は /nodebug です。

- /autoload

  /autoload=*dir* を使用すると、テストは ^UnitTestRoot ディレクトリのサブディレクトリ dir からロードされます。既定のサブディレクトリは _autoload です。

- /display

  /display=all を使用すると、メソッドの実行時に拡張情報が表示されます。/display=none を使用すると、限定された情報が表示されます。既定値は /display=all です。

userparam

呼び出し元から渡される任意の引数。既定では、マネージャは 1 つの値 /log を認識します。マネージャに、ターミナルへの出力を最小限に抑えて、<install-dir>/mgr ディレクトリの UNITTEST.LOG というファイルに詳細な結果を書き込むよう指示します。

4

%UnitTest の結果の表示

テストの結果を、次のいずれかの方法で表示できます。

- コンソール ― 基本的なテスト結果がコンソールに出力されます。
- %UnitTest.Result.TestAssert テーブル ― テスト結果が ^UnitTest.Result に表形式で保存され、%UnitTest.Result.TestAssert テーブルを介してアクセスできます。
- 管理ポータル ― ユニット・テストを実行すると、一連の Web ページで構成されるテスト・レポートが生成されます。テスト・レポートはネームスペース別に編成され、管理ポータルの [ユニットテスト・ポータル] 領域に表示できます。詳細は、"管理ポータルでの %UnitTest レポートの表示" を参照してください。

4.1 プログラムによる %UnitTest の結果の表示

テスト結果を含め、テストのアサーションは、構造化されたデータへのアクセスのために %UnitTest.Result.TestAssert テーブルにログ記録されます。このテーブルには以下のフィールドがあります。

Status

テストのアサーションの成功または失敗の値。可能な値は以下のとおりです。

テーブル 4-1:

論理値	意味
0	失敗
1	パス
2	スキップ

Action

テストの実行に使用された $$$AssertX マクロの名前。先頭の $$$ はテーブルには含まれません。

Description

$$$AssertX マクロに渡した test_description 引数の値。test_description を渡さなかった場合、このフィールドは既定で、$$$AssertX マクロの最初の引数の文字列表現になります。$$$AssertX マクロの引数の詳細は、"%UnitTest.TestCase クラスのマクロ" を参照してください。

Location

テストのアサーションが発生したテスト・クラス内の場所 (`label[+offset]^[|"ns"|]doc.ext` 形式)。

4.1.1 テストのアサーションの場所に関するトラブルシューティング

特定の状況では、テストのアサーションの場所がクラスに正しくマップされない可能性があります。例えば、すべての場所が生成された `INT` ルーチン内にある場合です。このような場合は、RunTest() の `qualifiers` 引数に `/keepsource` および `/generatemap` 修飾子を指定してテストを実行する必要があります。こうすることで、テスト・マネージャはルーチンの場所をソース・クラスに解決することができます。

4.2 管理ポータルでの %UnitTest レポートの表示

テストを実行すると、階層レポートが生成されます。このレポートには実行されたすべてのテストに関連する結果が含まれ、管理ポータルで参照できます。

レポートでテストは合格であると示されている場合、該当する $$$AssertX マクロは true を返し、テストで期待どおりの結果が得られたことを意味します。テストの失敗とは、マクロが false を返し、テストで期待どおりの結果が得られず、テスト対象のメソッドをデバッグする必要があることを意味します。

管理ポータルでレポートを表示するには、以下の手順を実行します。

1. %UnitTest クラスに USER ネームスペースの [ユニットテスト・ポータル] へのアクセス権を付与します。

```
USER>set $namespace = "%SYS"
%SYS>set ^SYS("Security","CSP","AllowPrefix","/csp/user/","%UnitTest.")=1
```

   注釈　セキュリティ上の理由により、このステップを 1 回実行しないと、管理ポータルの **[ユニットテスト・ポータル]** には移動できません。

2. 管理ポータルで、**[システムエクスプローラ]→[ツール]→[ユニットテスト・ポータル]** に移動して、USER ネームスペースに切り替えます。
3. [ユニットテスト・ポータル] を起動してテスト・レポートを表示するには、**[実行]** をクリックします。レポートが表示されます。
4. レポートのリンクをたどってドリルダウンし、より具体的な情報を見つけます。
   - 最初のページには、すべてのテスト・スイートの要約が表示されます。
   - 2 ページ目には、テスト・スイートごとの結果が表示されます。
   - 3 ページ目には、テスト・ケースごとの結果が表示されます。
   - 4 ページ目には、テスト・メソッドごとの結果が表示されます。
   - 最終ページには、テスト・メソッドで使用される各 `$$$AssertX` マクロの結果が表示されます。