記事
Toshihiko Minamoto · 2021年4月28日 5m read

ユニットテスト実行中のパフォーマンスデータの収集

数年ほど前、Caché Foundationsの講座（現「Developing Using InterSystems Objects and SQL」）において、%UnitTestフレームワークの基礎を講義していたことがあります。 その時、ある受講者から、ユニットテストを実行している間に、パフォーマンス統計を収集できるかどうかを尋ねられました。
それから数週間後、この質問に答えるために、%UnitTestの例にコードを追加したのですが、
ようやく、このコミュニティでも共有することにしました。

Processクラスの%SYSTEMには、プロセスについて収集可能なメトリクスが（所要時間以外に）いくつか提供されています。

- 所要時間
- 実行された行
- グローバル参照
- システムＣPU時間
- ユーザーＣPU時間
- ディスク読み取り時間

上記の統計を収集する**任意**
のユニットテストを有効にするには、%UnitTest.TestCaseのサブクラスを作成してプロパティを追加します。

```
Class Performance.TestCase Extends %UnitTest.TestCase
{
Property Duration As %Time;
Property Lines As %Integer;
Property Globals As %Integer;
Property SystemCPUTime As %Integer;
Property UserCPUTime As %Integer;
Property DiskReadTime As %Integer;
}
```

作成する特定のユニットテストは、%UnitTest.TestCaseではなく、新しいサブクラスから継承されている必要があります。

サブクラスではOnBeforeOneTest()を使って、各ユニットテストの統計データを初期化します。
DiskReadTime以外、現在値でプロパティを初期化します。

```
/// パフォーマンス統計を初期化します
Method OnBeforeOneTest(testname As %String) As %Status
{
 // 現在の値で初期化します
 set ..Duration = $zh
 set ..Lines = $system.Process.LinesExecuted()
 set ..Globals = $system.Process.GlobalReferences()
 set ..SystemCPUTime = $piece(CPUTime, ",", 1)
 set ..UserCPUTime = $piece(CPUTime, ",", 2)
 // ディスク時間を０にリセットし、カウントを開始します
 do $system.Process.ResetDiskReadTiming()
 do $system.Process.EnableDiskReadTiming()
```

```
  return $$$OK
}
```

OnAfterOneTest() を使って、各ユニットテストの統計データを確定します。
DiskReadTime以外、現在の値から初期値を減算します。

```
/// パフォーマンス統計を確定します
/// ここに、分析用にカウンターを別のテーブルに保存するためのコードを追加できます。
Method OnAfterOneTest(testname As %String) As %Status
{
  set ..Duration = $zh - ..Duration
  set ..Lines = $system.Process.LinesExecuted() - ..Lines
  set ..Globals = $system.Process.GlobalReferences() - ..Globals
   set CPUTime = $system.Process.GetCPUTime()
  set ..SystemCPUTime = $piece(CPUTime, ",", 1) - ..SystemCPUTime
  set ..UserCPUTime = $piece(CPUTime, ",", 2) - ..UserCPUTime
  // ディスク読み取り時間を取得し、カウントを停止します
  set ..DiskReadTime = $system.Process.DiskReadMilliseconds()
  do $system.Process.DisableDiskReadTiming()
  // ユニットテストログにメッセージを追加します

 set msg = "Performance: " _
  "Duration: " _..Duration _
  ", Lines: " _..Lines _
  ", Globals: " _..Globals _
  ", System CPU Time: " _(..SystemCPUTime / 1000) _
  ", User CPU Time: " _(..UserCPUTime / 1000) _
  ", Disk Read Time: " _(..DiskReadTime / 1000)
  do $$$LogMessage(msg)
  return $$$OK
}
```

ちょっとした技がもう1つあります。
統計を**収集するかしないか**を指定して、ユニットテストを実行したほうが良いかもしれません。
したがって、ユニットテストを呼び出すコードに引数（%Boolean
1または0）を追加し、何らかの方法で渡す必要があります。 テストを実際に実行するメソッド（RunTest()
またはほかのRun*() メソッドの1つ）は、第3引数に配列を取り、参照形式で渡します。
次は、そのサンプルコードです。

```
   // logging引数（1または0）を保持する配列を作成し、それを参照で渡します
  set p("logging") = logging
  do ##class(%UnitTest.Manager).RunTest(test, qualifiers, .p)
```

配列に渡す値は、OnBeforeOneTest() と OnAfterOneTest() でアクセスできます。
これを両方のメソッドの最初の行に追加します。

```
   if (..Manager.UserFields.GetAt("logging") = 0) { return $$$OK }
```

以上です！ 皆さんのご質問、コメント、その他のアイデアをぜひお聞かせください。

#Code Snippet #テスト #パフォーマンス #ベストプラクティス #Caché #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/%E3%83%A6%E3%83%8B%E3%83%83%E3%83%88%E3%83%86%E3%82%B9%E3%83%88%E5%AE%9F%E8%A1%8C%E4%B8%AD%E3%81%AE%E3%83%91%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%B3%E3%82%B9%E3%83%87%E3%83%BC%E3%82%BF%E3%81%AE%E5%8F%8E%E9%9B%86