Využitie používateľskej projekcie na Unit Testing

V predchádzajúcej časti sme sa oboznámili s balíčkom % UnitTesting a ukázali sme na praktickom príklade, ako s ním pracovať. Asi vám v priebehu čítania napadlo, či by nebolo možné tvorbu testovacích tried nejako automatizovať. Je to možné, dokonca to nie je nič zložité. Navyše sa ako bonus naučíme vytvárať používateľské projekcie. Ak ste zatiaľ s projekciami nepracovali, potom vedzte, že je to mechanizmus, ktorý Caché okrem iného používa pri tvorbe proxy tried zastupujúcich triedy Caché v prostredí hostiteľských technológií, ako sú C++ a Java či EJB. To však nie je zďaleka jediné možné použitie projekcie. A my vám jedno takéto netradičné použitie ukážeme.

Ako projekcia Caché funguje? Ak chceme aplikovať na našu triedu jednu alebo viac projekcií, jednoducho v rámci tela triedy pridáme deklaráciu **PROJECTION**. Kód obsiahnutý v triede implementujúcej projekciu (áno, projekcie sú obyčajné triedy) sa vykonáva v rámci každej kompilácie triedy obsahujúcej danú projekciu, a to na záver kompilačného procesu, teda po tom, čo sa vytvorí runtime kód a definície tabuliek (pri perzistentných triedach). Z toho vyplýva, že projekcia nemení kompilovanú triedu, ale môže robiť čokoľvek iné, napríklad generovať súbory či iné triedy.

Projekciu vytvoríme jednoduchým oddedením od bázovej triedy **% Studio.AbstractProjection**. Projekcia ponúka na implementovanie dve statické metódy: CreateProjection a RemoveProjection.

Prvá metóda sa vykoná po ukončení kompilácie triedy, druhá potom pred kompiláciou (presnejšie pred vygenerovaním runtime kódu) alebo pri zmazaní definície triedy.

A teraz jednoduchý príklad: Budeme implementovať pre jednoduchosť iba metódu **CreateProjection**. V tele metódy využijeme triedy v balíku **% Dictionary** na prechádzanie definíciou kompilovanej triedy a ku každej spustiteľnej metóde vytvoríme deklaráciu testovacej metódy v testovacej triede.

Kód je pomerne kompaktný, a tak tu uvedieme celú definíciu triedy projekcie.

```
Class tt.projection.UnitTest Extends %Projection.AbstractProjection
{

Parameter EXPORTFOLDER As STRING;

/// This method is called by the Class Compiler whenever an associated
/// class has finished compiling. classname contains the name of
/// the associated class while parameters contains an array of the
/// projection parameters subscripted by parameter name. The modified
/// for an incremental compile will contain a comma separated list of the methods that were
/// modified and it will be blank for a full compile

/// Any errors reported by this method are reported by the Class Compiler but
/// do not effect class compilation in any way.
ClassMethod CreateProjection(
        classname As %String,
        ByRef parameters As %String,
        modified As %String) As %Status
{
        #dim tMeth as %Dictionary.MethodDefinition
        // prejdi definiciu triedy a vytvor novu triedu (iba ak neexistuje) a prazdne testovacie metody (iba ak neexistuju)
        s len=$l(classname,".")
        s utclassname=classname_".UnitTest"

        if ##class(%Dictionary.ClassDefinition).%ExistsId(utclassname) {

s tUTClassDef=##class(%Dictionary.ClassDefinition).%OpenId(utclassname)
                        s msg="Modifikuji testovaci tridu "_utclassname
        } else {
                        s msg="Vytvarim testovaci tridu "_utclassname

s tUTClassDef=##class(%Dictionary.ClassDefinition).%New()

s tUTClassDef.Name=utclassname

s tUTClassDef.Super="%UnitTest.TestCase"
                        s tSC=tUTClassDef.%Save()
                        i $$$ISERR(tSC) w $System.Status.GetErrorText(tSC)
        }
        w !,"**********************************************************************"
        w !,msg
        w !,"**********************************************************************"

        // prejdi metody... - iba verejne
        s tClassDef=##class(%Dictionary.ClassDefinition).%OpenId(classname)
        s m="" f {
                        s tMeth=tClassDef.Methods.GetNext(.m) quit:m=""
                        if tMeth.Internal!(tMeth.Private) continue

s tUTMethName(0)="Test"_tMeth.Name_"OK"

s tUTMethName(1)="Test"_tMeth.Name_"NOK"
                        f i=0:1:1 {
                                if '##class(%Dictionary.MethodDefinition).%ExistsId(utclassname_"||"_tUTMethName(i)) {

s tUTMeth=##class(%Dictionary.MethodDefinition).%New()

s tUTMeth.Name=tUTMethName(i)

s tUTMeth.ClassMethod=0

s tUTMeth.parent=tUTClassDef
                                d tUTMeth.Implementation.WriteLine("        // TODO: dopsat parametery atd...")
                                d tUTMeth.Implementation.WriteLine("        // do ..Assert")
                                d tUTMeth.Implementation.WriteLine("quit")

s tSC=tUTMeth.%Save()

i $$$ISERR(tSC) w $System.Status.GetErrorText(tSC)
                                }
                        }
                }

        // potom vyexportuj do urceneho adresara
        s exportfile=$s($e(parameters("EXPORTFOLDER"),*)="\":parameters("EXPORTFOLDER"),1:parameters("EXPORTFOLDER")_"\")_utclassname_".xml"
        d $System.OBJ.Export(utclassname_".cls",exportfile)
        w !,"**********************************************************************"
        QUIT $$$OK
}

}
```

Teraz už stačí len pridať definíciu nami vytvorenej projekcie do triedy **tt.Tip18** z predchádzajúcej časti. To vykonáme jednoducho pridaním deklarácie

```
Projection UT As tt.projection.UnitTest(EXPORTFOLDER = "c:\temp");
```

Všimnite si, že naša projekcia zavádza parameter **EXPORTFOLDER**, do ktorého budeme exportovať testovacie triedy. A teraz už stačí iba triedu **tt.Tip18** skompilovať:

```
Compilation started on 03/02/2010 13:25:18 with qualifiers 'ckbu'
Compiling class tt.Tip18
Compiling routine tt.Tip18.1
**********************************************************************
Modifikuji testovaci tridu tt.Tip18.UnitTest
**********************************************************************
Exporting to XML started on 03/02/2010 13:25:18
Exporting class: tt.Tip18.UnitTest
Export finished successfully.

**********************************************************************
Compilation finished successfully in 0.093s.
```

Uvedený text je výpisom okna Output Studia pri kompilácii.

Malý kvíz na záver: Používa ZEN framework projekcie, alebo nie? V nasledujúcej časti sa budeme venovať práci s triedou % Installer.

DAN KUTÁČ, InterSystems

Ďalšie zaujímavé diely seriálu Tipy a triky s Caché, ktoré vychádzajú už od roku 2005, nájdete na stiahnutie tu: **http://www.intersystems.cz/education/ university/serialy/tipy-TrikyCache.html**

Intel možno presadí Light Peak cez Apple

Spoločnosť Intel sa chystá ponúknuť konkurenciu pre USB 3.0 vo forme optického prenosu dát cez Light Peak. Prvá generácia tejto technológie dosahuje prenosovú rýchlosť 10 Gb/s a vývojári ju sľubujú zrýchliť až na stovky gigabitov za sekundu. Light Peak je založený na prenose svetelného signálu cez optické vlákna s konvertormi na oboch stranách. Podľa spoločnosti Intel by táto technológia mohla slúžiť na pripojenie úložných zariadení, periférií, monitorov atď. Okrem rýchlosti Light Peak prináša aj výhodu dosahu, keďže optickým vláknam neprekážajú vzdialenosti do 100 metrov. Momentálne sa objavujú zvesti o tom, že presadiť túto technológiu pomôže spoločnosť Apple, ktorá by ju mohla použiť vo svojich zostavách už v budúcom roku. Keď bola v roku 2009 prvýkrát predstavená technológia Light Peak, bolo to práve na operačnom systéme Mac OS X od Apple. Teraz sa Steve Jobs vyjadril, že nebude podporovať USB 3.0, kým ho Intel nezabuduje priamo do svojich čipových súprav. Tým vzniká priestor pre Light Peak a práve Apple má vďaka svojej pozícii na trhu silu pretlačiť nový štandard.