# Dynamické odbavenie vlastností a metód tried

V tejto časti nášho seriálu sa oboznámime s veľmi užitočnou schopnosťou tried Caché, ktorá je k dispozícii už dlhší čas, ale až v posledných verziách sa dostáva viac do popredia, a to v súvislosti s dynamickým SQL a ZEN frameworkom. Ide o dynamické odbavenie (vyhodnotenie) vlastností a metód, po anglicky dynamic dispatch, inými slovami, o ošetrenie volania vlastností a metód, ktoré trieda nemá definované a nie sú súčasťou jej metaúdajov.

Všetci zaiste viete, že ak sa pokúsite pristúpiť za behu programu k neexistujúcej vlastnosti triedy alebo volať neexistujúcu metódu, program vám vráti výnimku. To je predvolené správanie Caché. Zaiste si však viete predstaviť situácie, keď toto správanie nie je žiaduce a vám by sa, naopak, hodilo, aby trieda nejakým spôsobom zareagovala bez toho, aby ste museli programovať spracovanie výnimiek.

Predstavte si napríklad situáciu, keď máte potrebné dve aplikácie alebo aj jednu viacvrstvovú, napríklad webovú aplikáciu. Na jednej strane hoci u webového klienta niečo urobíte a potrebujete preniesť dáta na stranu druhú – na server. Väčšinou ide o štruktúrované dáta, a teda k nim zostrojíme triedu, ktorá danú štruktúru opíše. No a toto budete chcieť robiť nie pre jednu webovú stránku, ale pre desiatky či stovky aplikačných stránok (obrazoviek). Iste sa vám nebude veľmi chcieť zakaždým definovať separátnu štruktúru (objekt). A práve v tomto okamihu prichádza neoceniteľný pomocník – dynamické odbavenie. Miesto mnohých špecializovaných tried, slúžiacich ako obyčajný kontajner na prenos dát z rôznych obrazoviek, si urobíte jednu jedinú triedu a tej implementujete kód, ktorý bude riadiť dynamické odbavenie jej vlastností. Tí z vás, ktorí používajú ZEN framework v novších verziách Caché, prípadne mali možnosť sa oboznámiť s novou verziou DeepSee, mohli naraziť na masívne využívanie triedy **% ZEN.proxyObject**. To je konkrétna implementácia dynamického odbavenia v praxi. Pomocou tejto triedy môžete veľmi elegantne prenášať objekty jazyka Javascript ako argumenty zenových metód (metód s kľúčovým slovom ZenMethod). Javascript totiž umožňuje definovať dynamické objekty, ktorých definície vznikajú za behu klienta (prehliadača) a tieto dynamické objekty môže Caché na strane servera spracovať práve vďaka implementácii dynamického odbavenia. API na dynamické odbavenie obsahuje nasledujúce metódy:

- %DispatchMethod(Method As %String, Args...)
- %DispatchClassMethod(Class As %String, Method As %String, Args...)
- %DispatchGetProperty(Property As %String)
- %DispatchSetProperty(Property As %String, Value)
- %DispatchSetMultidimProperty(Property As %String, Value, Subs...)

Názvy metód určujú ich použitie, je teda na vás programátoroch, čo budú jednotlivé metódy vykonávať. Viem si ľahko predstaviť, že takto môžete zostaviť objekt, ktorého inštancie budú ukladané nie v databáze, ale v textových súboroch na disku, prípadne môžete ľahko nad ľubovoľnou globálovou štruktúrou definovať objekt.

A teraz jedna malá, triviálne jednoduchá ukážka: Majme registrovanú triedu a tej implementujte metódu **% OnNew ()** tak, že jej argumentom bude zoznam párov názov: hodnota, oddelených dvojbodkou. Pomocou dynamického odbavenia môžeme jednoducho po inštancovaní triedy pracovať s odovzdanými pármi ako s vlastnosťami a ich hodnotami. Túto konštrukciu môžeme ľahko použiť ako základ komunikačného protokolu medzi našimi aplikáciami alebo našou a cudzími aplikáciami. Napríklad obalením webovými službami získame jednoduché a univerzálne objektové komunikačné rozhranie.

```
Class tt. Tip21 Extends  %RegisteredObject
{
property ValueNamePairsList as %String  [
Private, MutiDimensional  ];
□Method %DispatchGetProperty  (Property As
%String )
      {
            try    {
                     set    value= ..Value-
NamePairsList (Property)
            }   catch    (ex)   {
                   set     value=""
                     // nebo můžeme jedno-
duše vrátit runtime chybu pomocí
                     // throw
            }
           quit value
      }

   ///   Predame seznam paru  jmeno:hodnota
<BR>
   ///   <example>
   ///   USER>s  x=##class (tt.Tip21)  . %New
("jmeno:Josef:prameni:Novotny")
   ///   USER>w x.jmeno
   ///   USER>Josef
   ///   </example>
□Method %OnNew (initvalue As %CacheString) As
%Status [  Private, ProcedureBlock = 1, Serve-
rOnly = 1]
   {
              S i=1
              S len=$1 (initvalue, ":")
              For   {
                        Quit: i>len
                        s prop=$p (initvalue,
":", i)   s  i=i+1
                        s val=$p (initvalue,
":", i)   s  i=i+1
                        s ..ValueNamePairsList
(prop)=val
              }
             Quit $$$OK
     }
}
```

Ukážkový výstup:

```
SYM>s
x=##class(tt.Tip21).%New("jmeno:pepa:pri-
jmeni:novak")
SYM>w x.jmeno
pepa
SYM>w x.prijmeni
novak
```

Ak budete chcieť vidieť praktickú implementáciu **% ZEN.proxyObject**, poobzerajte sa v Zenovej komunite na Google, pred nejakým časom som tam umiestnil malú aplikáciu, ktorá dokáže zobrazovať projekty Caché alebo skupiny tried pomocou notácie UML. V jednej z tried projektu je trieda proxyObjekt využitá na odovzdávanie informácií z používateľskej obrazovky na spracovanie na server Caché. Odkaz na UML: **http://groups.google.com/group/intersystems-zen/web/app-uml-class-diagram-for-cache-ensemble?hl=en**

V budúcej časti sa budeme venovať novej implementácii dynamického SQL.

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/tipyTrikyCache.html**.

**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/tipyTrikyCache.html**