



Jython: Perzistentné objekty a Java projekcia

V predchádzajúcej časti sme ukázali, akým spôsobom sa robí pripojenie k databáze Caché. Teraz budeme pokračovať a budeme predpokladať, že už existujú triedy, ktoré majú v sebe zabudované rozhranie na perzistenciu ich inštancií. Takéto triedy možno vygenerovať v prostredí Caché pomocou postupu nazvaného Java projekcia. V princípe to funguje takto. V prostredí Caché sa nadefinujú perzistentné triedy, ktorých konkrétne inštancie chceme do databázy uložiť. Medzi týmito triedami môžu existovať rôzne komplikované väzby podobne ako v Jave. Dôležité je, že pri definícii tried sa v zdrojovom kóde uvedie i parameter Projection, definujúci zložku, do ktorej sa má vygenerovať zdrojový kód v Jave, majúci všetky metódy perzistentnej triedy Caché. Pri kompilácii triedy potom Caché vygeneruje kód použiteľný v prostredí JVM. Uvádzame ukážku takejto jednoduchej triedy:

```
Class JCP.Person Extends %Persistent
{
    Projection JavaProjection As
    %Projection.Java (ROOTDIR =
    "c:\Home\Java\CP");

    Property Name As %String;

    Property SSN As %String;

    ClassMethod populate(pCount As %Integer)
    As %Status
    {
        // tu sa nachadza kod na vytvaranie
        // instancii
        // . . . . .

        quit tsc
    }

    Method myToString() As %String
    {
        quit "Class: " .. %ClassName(1)
    }
}
```

Modul napísaný v Jythone potom definovanú triedu volá takto:

```
# import trid z balicka CacheDB.jar
from com.intersys.objects import *
from JCP import *

# definicia funkcie pracujucej s triedou Person
def workWithPerson():
    # vytvorenje instancie pripojenia ku
    Cache
    dbconnection=CacheDatabase.getData-
    base("jdbc:Cache://localhost:56773/DEV", "_sys-
    tem", "sys")

    # zavolam staticku metodu na vygene-
    rovanie instancie Person
    print "Generujem 5 instancii do db:"
```

```
Person.populate(dbconnection,5)

# vytvorim instanciu triedy Person
print "\nvytvaram instanciu rucne
a ukladam ju:"
person = Person(dbconnection)
person.setName("Doe, Joe A")
person.setSSN("111-12-1111")

# vypisem jej vlastnosti
print "\nziskam jej vlastnosti:"
print "Name: " + person.getName()
print "SSN: " + person.getSSN()

# ulozim instanciu do db
person.save()

# zavriem instanciu a spojenie s db
dbconnection.closeObject(person.getO-
ref())

dbconnection.close()

#
if __name__=="__main__":
    workWithPerson()
```

Výsledok je viditeľný ďalej:

```
C:\Home\Java\CP>jython CPTest03.py
Generujem 10 instancii do db:
Name: Eagleman,Brenda E.
Name: Lee,Ashley Y.
Name: Munt,Diane I.
Name: Ng,Barbara F.
Name: Umansky,Janice P.

vytvaram instanciu rucne a ukladam ju:
ziskam jej vlastnosti:
Name: Doe, Joe A
SSN: 111-12-1111
```

Z ukážky je zrejmé, že sme naraz získali silný nástroj, ktorý nielenže je schopný vďaka nástrojom v Caché vygenerovať použiteľné vzorky dát, ale dokáže i vytvoriť na základe definície perzistentnej triedy a tie vystaviť navonok a prístupníť ich prostrediu JVM a pomocou Jythonu veľmi jednoduchým a efektívnym spôsobom s týmito definíciami pracovať.

Pokiaľ sa nám podarilo uložiť inštancie triedy Person do databázy, očakávali by sme, že existuje i spôsob, ako sa k uloženým inštanciám dostať. Prvý a najjednoduchší spôsob využíva metódu open(), ktorá identifikuje uloženú inštanciu na základe jej vlastnosti zvanej object id.

Pozrieme sa teraz, ako možno pracovať s uvedenou triedou interaktívne za pomoci konzoly Jythonu. Z príkazového riadka si otvoríme konzolu Jythonu príkazom jython. Otvorí sa nám okno, do ktorého možno písať príkazy Jythonu, aké by sme písali do modulu. Rozdiel je v tom, že tu okamžite vidíme výsledok každého príkazu:

```
C:\Home\Java\CP>jython
Jython 2.2.1 on java1.6.0_04
Type "copyright", "credits" or "license" for
```

```
more information.
>>> from com.intersys.objects import *
>>> from JCP import *
>>>
>>> dbconnection=CacheDatabase.getDatabase("jdbc:
Cache://localhost:56773/DEV", "_system", "sys")
>>>
>>> id = Id(1)
>>> person = Person._open(dbconnection, id)
>>>
>>> print person.myToString()
Class: JCP.Person
>>>
>>> print "ID: ", person.getId()
ID: 1
>>>
>>> print "Name: ", person.getName()
Name: Doe, Joe A
>>>
>>> print "SSN: ", person.getSSN()
SSN: 111-12-1111
>>>
```

Toto je situácia, keď vieme, ktorú konkrétnu inštanciu požadujeme. Pokiaľ však potrebujeme pracovať s kolekciami inštancií, ktorých množinu môžeme vyjadriť efektívne dopytom SQL, môžeme postupovať nasledujúcim spôsobom:

```
>>> sql ="SELECT ID, Name, SSN FROM JCP.Per-
son WHERE Name %STARTSWITH ?"
>>>
>>> cq = CacheQuery(dbconnection, sql)
>>>
>>> selectionCond = "B"
>>>
>>> rs = cq.execute(selectionCond)
>>>
>>> rs.next()
1
>>> rs.getMetaData().getColumnCount()
3
>>>
>>> rs.getString(1)
'14'
>>> rs.getString(2)
'Bach,Mark Q.'
>>> rs.getString(3)
'805-84-9703'
>>>
>>>
>>> rs.next()
0
>>>
```

V prípade modulu potom výsledok môže vyzerať napríklad takto:

```
C:\Home\Java\CP>jython CPTest06.py
Conected.
Creating resultset with dynamic query.
SQL: SELECT ID, Name, SSN FROM JCP.Person
WHERE Name %STARTSWITH ?
Enter chars: B
Selected instances:
14: Bach,Mark Q.: 805-84-9703
# of instances: 1
```



■ ŠTEFAN HAVLÍČEK,
Sales engineer, InterSystems B. V.