Ontwikkelingshulp voor PL/SQL (2)

De stille kracht van Java

In een serie van drie artikelen geeft Lucas Jellema, technisch consultant bij AMIS, een overzicht van ideeën, patronen, richtlijnen en tools die PL/SQL ontwikkelaars kunnen overnemen uit de wereld van Java en J2EE. Hij gaat onder meer in op concepten uit de Java programmeertaal die een toepassing hebben in PL/SQL maar ook op open source-tools voor ondermeer logging, unit testen, automatiseren van batch-operaties en generatie van documentatie. Dit artikel is bestemd voor PL/SQL ontwikkelaars; kennis van Java is niet vereist. De source code voor dit artikel kan worden gedownload van http://www.amis.nl/tech_artikelen.php?id=158.

Een heel belangrijk concept bij Java applicatie-ontwikkeling, oorspronkelijk afkomstig uit de wereld van eXtreme Programming (zie www.extremeprogramming.org), is het zogenaamde (geautomatiseerde) unit testen. We zullen in dit artikel ingaan op een open source tool voor geautomatiseerd unit testen van PL/SQL programma's. Daarnaast kijken we naar Ant, een veelzijdig tool voor het uitvoeren van batch-operaties.

Unit testen

Kort gezegd is een unit test een test die een ontwikkelaar opstelt om zich ervan te verzekeren dat zijn of haar unit – procedure, functie of package – naar behoren functioneert. Een unit test is iets heel anders dan bijvoorbeeld een systeem: het laatste is gericht op het testen van functionaliteit en eigenschappen op applicatieniveau van de buitenkant, op een functionele manier. Een unit test richt zich op de kleinst mogelijke, zelfstandige, 'publieke' programma-eenheid. Om ontwikkelaars zover te krijgen dat ze grondiger gaan unit testen, is het noodzakelijk dat het proces zo simpel, snel en pijnloos mogelijk verloopt. De unit tests zijn een middel om de kwaliteit van de code te verhogen, maar ze zijn geen doel op zich. Een framework voor unit testen moet dus 'licht' zijn – gemakkelijk te installeren en te gebruiken, simpel om aan te passen en eenvoudig te runnen.

XP raadt aan dat je de unit tests tegelijk ontwikkelt met het ontwerpen van de code zelf, dus zelfs voorafgaand aan het eigenlijke programmeren. Het advies is verder om kleinere, modulaire programma's te schrijven die afzonderlijk getest kunnen worden en gezamenlijk een business service implementeren. Een unit test voert een test uit op een individueel programma – een enkele PL/SQL procedure of function, al dan niet binnen een package. Omdat het ontwerpen van de code hand in hand gaat met het ontwikkelen van de unit test-cases, word je gedwongen om het ontwerp beter door te denken en van meerdere kanten te bekijken. Programma-ontwerp en unit test ontwikkeling zijn een iteratief proces. Eenmaal afgerond kan de daadwerkelijke code geschreven worden op basis van de specificaties in het ontwerp, en op ieder moment getest. De unit test biedt een duidelijk fundament voor de ontwikkelaar, of de code nieuw geschreven wordt of in een onderhoudsfase onderhanden wordt genomen. De unit test biedt op ieder moment directe feedback op de correctheid van de code.

Fluitje van een cent

Als de unit tests met een generieke interface worden geschreven, kan het uitvoeren van een test – of zelfs een suite van tests – opgezet worden als geautomatiseerd proces. Op dat moment zal het volledig en grondig unit testen van alle onderdelen van een applicatie een fluitje van een cent zijn. Althans, in theorie.

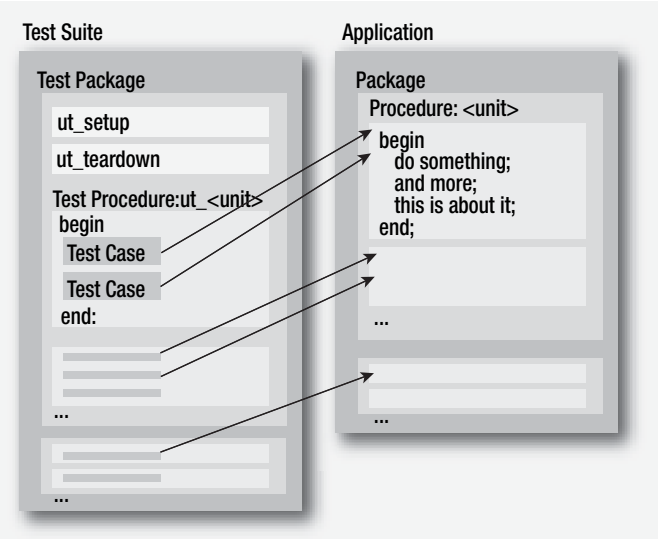
Unit tests moeten niet alleen maar controleren of een programma succesvol aangeroepen kan worden. Het moet diverse situaties testen, verschillende combinaties van waarden van input-parameters, ook waarden die exceptions opleveren. De test moet verifiëren dat het programma de juiste resultaten oplevert bij correcte invoerwaarden, maar ook goed omgaat met foutieve invoer. Overigens kan het wel nodig zijn dat een unit test verder kijkt dan alleen naar de out-parameters en function-return waarde van het aangeroepen programma. Het kan goed zijn dat de te testen procedure een database manipuleert, een file schrijft of bijvoorbeeld een e-mail verstuurt. De unit test zal ook dat soort 'bijwerkingen' moeten controleren.

Testresultaten moeten helder en eenduidig zijn: de test is succesvol of is mislukt.
De Java wereld kent het JUnit framework voor unit testen. Dit is een van de meest bekende en meest gebruikte open-source frameworks (zie http://junit.org/index.htm). Het biedt een infrastructuur voor geautomatiseerd unit testen van Java code. In aanvulling op JUnit zijn ook frameworks als HttpUnit (voor functionele tests van webapplicaties) en Cactus (voor unit testen van J2EE applicaties) beschikbaar gekomen. Voor PL/SQL ontwikkelaars is er utPLSQL, een open source framework voor unit testen van PL/SQL code. utPLSQL is ontwikkeld onder leiding van Steven Feuerstein – een heel bekende naam op het gebied van PL/SQL en auteur van een achttal zeer populaire boeken over PL/SQL. Het utPLSQL framework beschrijft een proces en biedt een verzameling packages die PL/SQL ontwikkelaars kunnen gebruiken om hun code grondig, efficiënt en eenvoudig te unit testen.

Karakteristieken van utPLSQL

PLSQL

Homepage	http://utplsql.sourceforge.net/
Bestaat sinds	Juni 2000
Meeste recente release	2.1.1 (17 Juni 2003)
Status	Productie
Onderliggende technologie	PL/SQL
Ontwikkelteam	10
Zip-grootte	300 Kb
Gerelateerde projecten	OUnit (http://www.ounit.com) en QNXO (http://www.qnxo.com/)



Figuur 1. De procedures in utPLSQL implementeren elk een unit test, behorend bij precies één procedure in de applicatie

Hoe werkt het?

Met utPLSQL kun je (PL/SQL) test-packages maken met daarin procedures. Deze procedures implementeren elk een unit test – deze behoort bij precies één procedure in de testapplicatie. Een testprocedure bevat één of (meestal) meerdere test-cases, die elk kunnen slagen of falen. Een test-case ziet er in grote lijnen als volgt uit:

- Initialiseer variabelen
- Roep de testprocedure aan
- Verifieer de resultaten – zowel de function return-waarde en de out-parameters als ook de bijwerkingen zoals database en e-mail – en registreer de bevindingen van de unit test.

De derde stap wordt uitgevoerd met behulp van het utAssert package. Dit package biedt twee faciliteiten: het controleert of de opgegeven 'assertion' – aanname - klopt en het registreert de conclusie in een centraal register binnen het utPLSQL framework. Package utAsserts ondersteunt een groot aantal soorten assertions, zoals: gelijkheid en ongelijkheid, IS NULL, het maken van een exception. Het test ook de Boolean conditie voor diverse datatypes zoals strings, number, files, pipes, collections. Nadat alle unit tests zijn uitgevoerd en alle bevindingen zijn vastgelegd door utAssert, worden de conclusies gerapporteerd.
Test-packages kunnen bij utPLSQL aangemeld worden, individueel of als onderdeel van een testsuite. Je kunt utPLSQL een complete testsuite laten uitvoeren of een enkel, specifiek test-package. Het utPLSQL framework vindt alle testprocedures en roept ze één voor één aan[1]. Deze procedures kunnen worden gebruikt voor huishoudelijke taken: in ut_setup kun je initialisatie voor het gehele test-package uitvoeren, inclusief zo nodig de creatie van database objecten of data. In ut_teardown kan je overblijfselen van de tests weer opruimen, om te garanderen dat het runnen van de test geen blijvende resultaten heeft. Een aardige extra voorziening in utPLSQL is de mogelijkheid om een test-package – met stub-code – te genereren voor ieder bestaand package in de applicatie waarvoor je unit tests wilt creëren.

Aan de slag

De TestSuite is opgebouwd uit TestPackages met een testprocedure voor iedere program unit. We gaan als volgt te werk:

- Download utPLSQL van http://oracle.oreilly.com/utplsql.
- Unzip en installeer in een database (7.3.4 of hoger).
- Ontwerp de unit tests en de test-cases: specificeer welke programma's moeten worden getest, wat de relevante

1 Een test-package moet ook de procedures ut_setup en ut_teardown bevatten, ook al bevatten ze geen andere code dan begin null; end;

combinaties van waarden van input-parameters zijn en wat de verwachte uitkomsten zijn voor elk van die combinaties.
- Ontwikkel de test-packages volgens de utPLSQL voorschriften: met de procedures ut_setup en ut_teardown en een procedure ut_<PROGRAM_UNIT_TO_TEST> voor iedere procedure die moet worden getest.
- Registreer de test-package in utPLSQL (eventueel als onderdeel van een test suite)
- Laat utPLSQL de test-package of de test suite uitvoeren

Laten we utPLSQL nader onderzoeken aan de hand van een eenvoudig voorbeeld. Het package APP_ARITHMETIC bevat function add (a in number, b in number). Deze geeft de som van de twee input-parameters terug. Als een input-parameter NULL bevat, wordt de parameter in deze functie geïnterpreteerd als 0. De test-cases voor deze functie zijn te zien in tabel 1: er zijn geen bijwerkingen.

```
<kopregel> Input Parameter 1   Input Parameter 2     Resultaat
x (een willekeurige numerieke waarde)     Y (een willekeurige numerieke
waarde)   x+y
x (een willekeurige numerieke waarde)     NULL       x
NULL      y (een willekeurige numerieke waarde)      y
NULL      NULL      0
```

In dit geval werk ik het test-package ut_APP_ARITHMETIC met de hand uit, en wel als volgt:

```
create or replace package body ut_APP_ARITHMETIC
as

procedure ut_setup
is
begin
 null;
end;

procedure ut_teardown
is
begin
 null;
end;

procedure ut_add
is
begin
 -- test case met gewone numerieke parameter-waarden
 utassert.eq
 ( 'Normale input-parameters (5 en 8) '
 ,  APP_ARITHMETIC.add( 8, 5)
 ,  (5+8)
 );
 -- test case met de eerste parameter NULL
 utassert.eq
 ( 'Eerste input parameter is NULL'
 ,  APP_ARITHMETIC.add( NULL, 5)
 ,  5
 );
 -- test case met tweede parameter NULL
 utassert.eq
 ( 'Tweede input parameter is NULL'
 ,  APP_ARITHMETIC.add( 12, NULL)
 ,  12
 );
  -- test case met beide parameters NULL
 utassert.eq
 ( 'Beide input-parameters zijn NULL'
 ,  APP_ARITHMETIC.add( NULL, NULL)
 ,  0
 );
end;

end;
```

Nu we de unit test ontwikkeld hebben kunnen we volgens de XP methode aan de slag met het schrijven van de echte code:

```
CREATE OR REPLACE package body app_arithmetic
as

function add( a in number, b in number)
return number
is
begin
  return nvl(a,0) + b;
end;

end;
```

We kunnen het test-package registreren als onderdeel van een test-suite, maar we kunnen het ook direct laten uitvoeren:

```
begin
  utPLSQL.test(package_in=>'APP_ARITHMETIC', recompile_in=>false, owner_
in => 'APP_SCHEMA');
end;
```

De testresultaten zien eruit als in Figuur 2. Vervolgens corrigeren we de code:

```
CREATE OR REPLACE package body app_arithmetic
as

function add( a in number, b in number)
return number
is
begin
  return nvl(a,0) + nvl(b,0);
end;

end;
```

Figuur 2. Met de test-case wordt eerst de bug geverifieerd

Figuur 3. Nadat de code is aangepast, kun je de test-case gebruiken om de oplossing te verifiëren

Wanneer we de test vervolgens opnieuw uitvoeren zijn de resultaten wat geruststellender[2] (zie Figuur 3). Stel nu dat we al bestaande packages hebben. We kunnen dan gebruik maken van de Generate Test Package faciliteit van utPLSQL om voor geselecteerde packages de kale test-packages te genereren. De gegenereerde test-packages bevatten stubs voor iedere procedure en function in het originele package, waar je zelf de test-case(s) kunt uitwerken. Het volgende statement is voldoende voor het genereren van het skelet van een test-package:

```
begin
  utGen.testpkg ( package_in => 'APP_ARITHMETIC', SCHEMA_IN=>'APP_
SCHEMA');
end;
```

De test-cases kunnen ook al direct worden meegegenereerd door utPLSQL; je kunt test-cases op de volgende manier definiëren:

```
begin
  utGen.testpkg_from_string
  ( package_in => 'APP_ARITHMETIC'
  , SCHEMA_IN=>'APP_SCHEMA'
  , grid_in =>
'Add||Test Case 1|Normale input-parameters (5 en 8)|5;8|13|EQ
Add||Test Case 2|Eerste input parameter is NULL|5;|5|EQ
Add||Test Case 3|Tweede input parameter is NULL|;8|8|EQ
Add||Test Case 4|Beide input parameters zijn NULL|;|0|EQ'
  );
end;
```

Hieronder een deel van de door utPLSQL gegenereerde code:

```
CREATE OR REPLACE PACKAGE BODY ut_APP_ARITHMETIC
IS
PROCEDURE ut_setup
IS
BEGIN
NULL;
END;
PROCEDURE ut_teardown
IS
BEGIN
NULL;
END;
-- For each program to test...
PROCEDURE ut_ADD
IS
-- Verify and complete data types.
against_this NUMBER;
check_this NUMBER;
BEGIN
-- Define "control" operation for "Test Case 1"
against_this := 13;
-- Execute test code for "Test Case 1"
check_this :=
APP_ARITHMETIC.ADD (
A => 5
,
B => 8
);
-- Assert success for "Test Case 1"
-- Compare the two values.
utAssert.eq (
'Normale input-parameters (5 en 8)',
check_this,
against_this
);
-- End of test for "Test Case 1"
-- Define "control" operation for "Test Case 2"
against_this := 5;
-- Execute test code for "Test Case 2"
check_this :=
APP_ARITHMETIC.ADD (
A => 5
,
B => NULL
);
-- Assert success for "Test Case 2"
-- Compare the two values.
utAssert.eq (
'Eerste input parameter is NULL',
check_this,
against_this
);
-- End of test for "Test Case 2"
....
```

2 De resultaten worden door utPLSQL standard naar dbms_output gestuurd. Ze zijn ook beschikbaar via het utResult package. Je kunt een eigen applicatie ontwikkelen – met bijvoorbeeld Oracle Forms of Java - van waaruit je unit tests kan runnen en de resultaten kan bekijken (zie de passages in dit artikel over OUnit, een voorbeeld van een dergelijke applicatie).

Food for thought

Als iemand een bug rapporteert, is je eerste ingeving om de code in te duiken om het probleem te vinden en op te lossen. Maar er is een betere, meer gestructureerde en tot hogere kwaliteit leidende aanpak. Voordat je in de code begint te wroeten, schrijf je eerst een testcase die de bug verifieert, dat wil zeggen: waarmee je het gerapporteerde, incorrecte gedrag kan constateren; deze test zal dan ook falen. Ga dan pas de code in, analyseer deze en bepaal de oplossing. Als deze is geïmplementeerd kun je de testcase gebruiken om de oplossing te verifiëren - deze keer zou de test niet mogen falen.

De nijvere mier

Ant (Another Neat Tool) is een door Java-ontwikkelaars veelgebruikt open source tool. Ant is een simpel hulpmiddel met heel nuttige functionaliteit. Ant is een geautomatiseerd build-en-deploy tool. Ant kan een grote verscheidenheid aan taken uitvoeren, taken die waardevol zijn bijvoorbeeld bij het build-en-deploy proces van Java of PL/SQL applicaties. Standaardtaken zijn ondermeer het creëren en verwijderen van directory's, het kopiëren en downloaden van files, FTP-operaties, creëren en extraheren van archieven (zip- en jar-files) en het genereren van documentatie (daarover meer in het volgende artikel in deze reeks). Onder de meer geavanceerde features vallen het uitchecken van source code uit versiebeheersystemen zoals CVS en SourceSafe, het uitvoeren van SQL of PL/SQL scripts, het transformeren van XML documenten naar bijvoorbeeld PDF of HTML en het genereren van DDL Scripts en Oracle Forms uit Oracle Designer. Voor dat laatste heeft Oracle Consulting de tool J-RAG ontwikkeld (een introductie van dit tool is te vinden op http://technology.amis.nl/blog/index.php?p=164& page=5).

Karakteristieken van Apache Ant

Homepage	http://ant.apache.org/
Bestaat sinds	Januari 2000
Meeste recente release	1.6.2 (Juli 2004)
Status	Productie
Onderliggende technologie	Java, XML
Ontwikkelteam	25
Zip-grootte	9.5 Mb
Gerelateerde projecten	AntContrib (http://ant-contrib.sourceforge.net/) , Antelope (http://antelope.tigris.org/), AntHill (http://www.urbancode.com/projects/anthill/) en een heleboel andere

De unit test biedt op ieder moment directe feedback op de correctheid van de code

Ant heeft een aantal eigenschappen die het doen onderscheiden van gewone bat-files en shell-scripts:

- Simpele, gestructureerde, op XML gebaseerde syntax.
- Krachtige, herbruikbare acties (generieke taken voor file systeem, database, en applicatieserver).
- Geïntegreerd met verschillende ontwikkelomgevingen zoals JDeveloper en Eclipse.
- Portable tussen verschillende platformen en operating systems; niet meer het onderscheid tussen.bat en .sh scripts et cetera.
- Individuele taken kunnen worden geselecteerd en uitgevoerd maar ook complexe operaties die uit tientallen of honderden van elkaar afhankelijke taken bestaan.
- Taken kunnen parallel aan elkaar uitgevoerd worden.

Ant wordt meestal met Java projecten geassocieerd. Maar Ant – hoewel Java- en XML-gebaseerd – kan alle taken waarvoor een generieke implementatie is ontwikkeld – en dat zijn er verschrikkelijk veel – uitvoeren, en zo ook voor SQL en PL/SQL ontwikkeltrajecten:

- Nachtelijke build (opschonen van de omgeving, aanmaken van directory's, prepareren van de database, ophalen van sources uit het versiebeheersysteem, eventuele generatie van programmacode, compileren en eventueel kopiëren of archiveren)
- Het uitvoeren van geautomatiseerde unit tests
- Genereren van documentatie
- Controle van programmeerstandaarden
- Beheer van testdata (opschonen en laden van schone testdata)
- Uitrol van applicatie (genereer installatie-script, bundel alle code, enzovoort)

Ant wordt aangestuurd door een build.xml file. Deze xml- file bevat property-definities en taken (<target> elementen). Een taak wordt geïmplementeerd door een Java class[3] die een bepaalde geparametriseerde operatie uitvoert. Zodra Ant stand-alone geïnstalleerd is kan een Ant-build-script vanaf de command-prompt uitgevoerd worden:

3 Deze Java class is in de meeste gevallen al geschreven en meegeleverd met Ant. Als Ant-gebruiker heb je alleen met de XML file te maken, niet met Java code.

```
ant –buildfile myBuildFile.xml
```

Gemakkelijker nog is het gebruik van bijvoorbeeld JDeveloper 10g: je creëert of opent een Ant build.xml file en vanuit het rechtermuisknop-menu kun je één of alle taken laten uitvoeren. Een erg krachtige Ant-task voor onze situatie is de sql-task. Deze biedt ons de mogelijkheid om SQL en PL/SQL operaties in batch te laten uitvoeren. Met deze taak kunnen we bijvoorbeeld onze database installatie scripts in Ant gaan ontwikkelen. Een aantal zeer nuttige tips over het gebruik van deze task vind je op: http://www.bris.ac.uk/is/projects/portal/team/lilian/AntSql. Een voorbeeld van het gebruik van de sql-task om een SQL script uit te voeren staat hieronder. Deze taak roept het script create_triggers.sql script in de directory /app/build/ddl/sql aan en voert het uit binnen de aangegeven database-connectie:

```
<target name="sql">
  <sql driver="oracle.jdbc.driver.OracleDriver"
       url="jdbc:oracle:thin:@amis10gDB7:1521:orcl"

       userid="scott"
       password="tiger">
    <fileset dir="/app/build/ddl/sql">
      <include name="create_triggers.sql"/>
    </fileset>
  </sql>
</target>
```

Je kunt ook één of meer SQL- of PL/SQL-statements rechtstreeks in de task opnemen. Een voorbeeld daarvan is de volgende task die de utPLSQL unit test uitvoert die we eerder in dit artikel hebben besproken:

```
<target name="unitTest">
<sql
driver = "oracle.jdbc.driver.OracleDriver"
url = "${database.jdbc.url}"
userid = "${database.user}"
password = "${database.password}"
print = "true"
>
  utPLSQL.test(package_in=>'APP_ARITHMETIC', recompile_in=>false, sche-
ma_in=> 'APP_SCHEMA');
</sql>
</target>
```

Met ${..} wordt verwezen naar property's die in een externe property's file worden gedefinieerd. Het voordeel van die property's-file is dat deze de enige is die voor de doelomgeving hoeft te worden aangepast; het centrale build-script hoeft niet te worden bewerkt. Meer nuttige Ant-tasks worden onder meer op OTN besproken.

Vooruitblik

In het volgende artikel van deze serie zullen open source tools voor de generatie van documentatie voor PL/SQL code en Oracle Forms aan de orde komen. Daarnaast zullen we kijken naar de betekenis van JDeveloper als ontwikkelgereedschap voor SQL en PL/SQL development.

Resources

- Artikel over unit testing en de eXtreme Programming. Achtergronden: http://oracle.oreilly.com/pub/a/oreilly/oracle/utplsql/news/fulldoc.html
- Homepage van utPLSQL: http://utplsql.sourceforge.net/ en Ounit: http://www.ounit.com/
- Homepage van Ant: http://ant.apache.org/
- J-RAG – tool voor in batch met Ant vanuit Oracle Designer genereren van Oracle Forms en DDL scripts; zie: http://technology.amis.nl/blog/index.php?p=164&page=5
- Tips over het gebruik van de ANT sql-task: http://www.bris.ac.uk/is/projects/portal/team/lilian/AntSql. Zie ook: http://ant.apache.org/manual/CoreTasks/sql.html voor de achterliggende documentatie.
- Meer nuttige ANT-tasks in een Oracle ontwikkelomgeving: http://otn.oracle.com/oramag/oracle/02-nov/o62odev_ant.html en http://www.oracle.com/technology/oramag/oracle/03-jan/o13ant.html
- Source code bij het artikel: http://www.amis.nl/tech_artikelen.php?id=158

Lucas Jellema
is sinds 1994 werkzaam met Oracle- en later ook Java-technologie, eerst bij Oracle en sinds 2002 bij AMIS Services B.V. in Nieuwegein in de rol van Technisch Consultant. Met collega's onderhoudt hij een weblog (http://technology.amis.nl/blog/) rondom Oracle- en Java-technologie. Voor verdere vragen kan hij worden bereikt per e-mail: jellema@amis.nl.