Open Sesame Java 5

Eindverslag ter afronding van bovengenoemd project, zoals overeengekomen in de gelijknamige overeenkomst tussen Stichting NLnet en Aduna b.v.

Inleiding

Met de introductie van Java 5[1] zijn er een aantal nieuwe mogelijkheden aan de programmeertaal toegevoegd die erop gericht zijn om de ontwikkeling van software te vereenvoudigen, de kans op fouten te verminderen en het resultaat expressiever te maken. Naar aanleiding van een online poll waarin een grote meerderheid van de community voor een overstap naar Java 5 bleek te zijn, zijn we in Sesame 2.0-alpha 2 met deze overstap begonnen. Dit document doet verslag van de werkzaamheden die hiermee gepaard zijn gegaan en geeft een indruk van het bereikte resultaat.

Java 5: de nieuwe mogelijkheden

De belangrijkste nieuwe mogelijkheden in Java 5 zijn ongetwijfeld: *Generics*, *Enhanced for Loops*, en *Typesafe Enums*. *Generics* is een mechanisme om code te schrijven dat gebruik maakt van abstracte object types, min of meer vergelijkbaar met "templates" in C++. De *Enhanced for Loop* is een syntactische uitbreiding om het itereren over een verzamelingen objecten, een veel voorkomende operatie, te vereenvoudigen en de kans op fouten hierin te verkleinen. *Typesafe Enums* bieden de mogelijkheid om op een veilige, object-geörienteerde manier een vaste set van waardes te representeren, waar voorheen vaak constanten met een integer waarde gebruikt werden. Meer gedetailleerde informatie over deze en andere uitbreidingen in Java 5 is te vinden in *New Features and Enhancements J2SE 5.0*[2].

De overstap naar Java 5

Volgens het vooraf opgestelde plan van aanpak bestonden de werkzaamheden grofweg gezien uit twee hoofdtaken:

(a) Het her-ontwerpen/omschrijven van de publieke APIs om gebruik te maken van *Generics* en *Typesafe Enums*.
(b) Het omschrijven van de interne code om gebruik te maken van Java 5 features in het algemeen, o.a. *Enhanced for Loops*.

Het zwaartepunt van het werk lag duidelijk bij de eerste taak; afgezien van de *Enhanced for loop* en *Autoboxing/Unboxing* hebben alle nieuwe features op de een of andere manier invloed op APIs. Verder bleek het handiger te zijn om het werk in een iteratief proces uit te voeren, waarbij de APIs met alle bijbehorende code een voor een werden aangepakt. De geschatte inspanning van twee mensmaanden voor het geheel bleek redelijk te kloppen: taak a heefts iets meer dan één mensmaand gekost, taak b iets minder. De ondersteuning vanuit Eclipse als ontwikkelplatform voor het omzetten van interne code heeft hierbij duidelijk geholpen om taak b redelijk snel af te ronden.

Bereikte resultaten

Duidelijkere/expressievere APIs
Het toepassen van *Generics* heeft geleid tot duidelijkere APIs omdat deze expressiever zijn geworden. Waar voorheen een methode `List` als return type had en uit de begeleidende

1 http://java.sun.com/j2se/1.5.0/index.jsp
2 http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html

documentatie moest blijken welke type objecten in deze lijst zitten, kan deze methode dit nu precies specificeren, bijvoorbeeld `List<Value>` voor een lijst waar `Value` objecten in zitten. Hetzelfde geldt uiteraard ook voor parameters van methodes.

Ook staat Java 5 nu toe dat een methode een specifieker return type heeft dan wat er in een superclass of interface gedefinieerd is. In Sesame is dit onder andere gebruikt voor de class `RStatement`, welke interface `Statement` implementeerd. De laatste definieert methodes als `getSubject()`, `getPredicate()` en `getObject()`, welke `Value` object teruggeven. In het geval van `RStatement` zijn dit echter altijd `RValue` objecten en dankzij de uitbreidingen in Java 5 kan dat ook als zodanig gedeclareerd worden. Voorheen voegden we meestal extra methodes toe als `getRSubject()` om dit duidelijk te maken en om type casting te voorkomen.

Verbeterde APIs
Niet alleen zijn de APIs duidelijker en expressiever geworden, ze zijn hier en daar ook verbeterd. De introductie van *Varargs* in Java 5 zorgt ervoor dat er nu ook methodes geschreven kunnen worden die 0-of-meer argumenten van hetzelfde type accepteren. Een mooi voorbeeld hiervan is de methode om een `Value`-tuple aan een `TupleSet` toe te voegen. `TupleSet` heeft hier nu twee methodes voor: een met een lijst van `Values` en een met een variabel aantal `Value` argumenten: `addTuple(Value... values)`.

Een andere verbetering is het beperken van oneigenlijk gebruik van interfaces voor de definitie van constanten. Door constanten in een interface te definiëren en een class als uitbreiding van deze interface te definiëren kunnen de betreffende constanten eenvoudiger binnen deze class gebruikt worden. Dit voegt echter wel een oneigenlijk type toe aan de class hierarchie, iets wat soms wordt aangeduid als een “Constant Interface antipattern“. In Sesame werd deze constructie o.a. gebruikt om namen van XML tags op een centrale plek te definiëren. Zowel de parser als de writer voor het betreffende bestandsformaat implementeerden dan deze interface. Deze constructies zijn nu vervangen door *Static Imports*, wat een schonere, en dus duidelijkere, class hierarchie oplevert.

Stabielere code
Het omschrijven van Sesame naar Java 5 heeft ook stabielere code opgeleverd. Het eerder genoemde *Generics* mechanisme zorgt ervoor dat typerings fouten vaker al door de compiler gedetecteerd worden en dat potentiële verkeerde type casts minder vaak nodig zijn.

Typesafe Enums bieden een mechanisme om eenvoudig een set van constanten te definiëren, zoals bijvoorbeeld een set van RDF bestandsformaten (RDF/XML, N-Triples, Turtle, etc.), en hieraan te refereren met een type-naam. Voorheen werden zulke constanten vaak gedefinieerd als integer waardes, maar het gebrek aan semantiek ervan leidt snel tot onduidelijke code en is een potentiële bron van fouten. Een methode die gedefinieerd is als `evaluateQuery(QueryLanguage, String)` is eenvoudiger te begrijpen en te gebruiken dan het equivalent met integer constanten: `evaluateQuery(int, String)`. Deze laatste variant kent daarnaast het gevaar dat er een compleet ongerelateerde constante als parameter wordt meegegeven.

Enhanced for Loops, ten slotte, bieden een versimpelde syntax voor het itereren over een verzameling objecten. De compiler zorgt er hierbij voor dat over de verzameling geïtereerd wordt zodat alleen de operatie op de individuele objecten nog gecodeerd hoeft te worden. Hiermee wordt het gevaar van oneindige loops door verkeerd gebruik van iterators en index variabelen voorkomen en wordt de code vereenvoudigd. In geval van geneste for-loops wordt ook het foutief hergebruik van index variabelen voorkomen.

Kleinere codebase

Door het gebruik van *Generics* konden een redelijk aantal specifieke classes en interfaces worden vervangen door generieke alternatieven. Omwille van de schaalbaarheid naar grote hoeveelheden data wordt er in Sesame veelvuldig gebruik gemaakt van iterators. Hiervoor had Sesame interfaces zoals `ResourceIterator`, `ValueIterator` en `StatementIterator` met methodes als `hasNext()` en `next()` voor het itereren over de objecten, en `close()` voor het vrijgeven van de door de iterator gebruikte resources. Verder waren er utility classes voor elk van deze interfaces als `EmptyValueIterator`, `SingleValueIterator`, `ValueCollectionIterator`, etc. Al deze interfaces en classes zijn nu vervangen door een getypeerde `CloseableIterator`, een extensie van de standaard `java.util.Iterator`, en generieke utility classes.

Door het vervangen van de specifieke classes en interfaces door generieke alternatieven is de codebase van Sesame significant verkleind, wat een aantal voordelen heeft. Ten eerste maakt het Sesame nog beter geschikt voor gebruik in een web-omgeving (Sesame was al min of meer favoriet vanwege de beperkte omvang van de codebase). Ten tweede betekend een kleinere codebase ook dat er minder te leren en te onthouden is voor ontwikkelaars, wat gebruik van Sesame als library vereenvoudigd.

Evaluatie van het resultaat

Nu de overstap naar Java 5 afgerond is, hebben we sterk het gevoel dat dit een goede zet is geweest. Naarmate het werk vorderde ontdekten we steeds meer mogelijkheden om de Sesame code te verbeteren. Een initiële zorg dat we met deze stap het gebruik van Sesame 2.x in omgevingen die nog niet over zijn gegaan naar Java 5 onmogelijk maakten is door een lid van de community weggenomen: met behulp van Retroweaver[3] bleek het mogelijk te zijn om de nieuwe Sesame code terug te vertalen naar Java 1.4 compatibele code[4]. Ontwikkelaars die (nog) niet over kunnen naar Java 5 kunnen zodoende toch gebruik maken van de laatste ontwikkelingen in Sesame. De noodzaak om de Sesame 1.x versies te blijven onderhouden wordt hierdoor aanzienlijk verminderd.

Inmiddels zijn er twee alpha releases geweest met daarin veranderingen die samenhangen met de overstap naar Java 5. Recent zijn er nog wat verfijningen aangebracht in de code, mede naar aanleiding van feedback van gebruikers. Deze verfijningen zullen in de eerstvolgende release terug te vinden zijn. (Informatie over) deze releases is, zoals altijd, te vinden op de website voor het project: http://www.openrdf.org/.

3 http://retroweaver.sourceforge.net/
4 http://simile.mit.edu/mail/ReadMsg?listName=General&msgNo=1679