Core

Veel ontwikkelaars die iets van Groovy hebben gezien, zijn hier erg enthousiast over. En terecht ook! Met Groovy schrijf je in veel situaties met veel minder code veel meer functionaliteit. Ondanks het enthousiasme kunnen veel ontwikkelaars Groovy echter niet gebruiken binnen hun dagelijks werk, omdat het niet past binnen de gebruikte Java-omgeving. Dit artikel laat zien hoe je op een zinvolle manier Groovy binnen Java-projecten kunt gebruiken.

Real life Groovy

Voordelen taal ook bruikbaar in Java-projecten

Onterecht wordt Groovy vaak als synoniem gezien voor het webframework Grails. Dat is raar, want Groovy is een taal net zoals Java dat is en Grails is een webframework gebaseerd op Groovy. De reden dat veel ontwikkelaars Groovy direct associëren met het snel en eenvoudig ontwikkelen van webapplicaties komt doordat Grails een enorme boost aan de bekendheid van Groovy heeft gegeven. Grails is dan ook een heel krachtig framework. Het framework drukt (net zoals veel andere frameworks) echter wel een enorme stempel op de architectuur van een applicatie. Dat kan prima werken voor nieuwe projecten, maar is vaak niet bruikbaar in combinatie met een bestaand project met bestaande code. Er zijn al veel introducerende artikelen over Groovy te vinden, onder andere in het Java Magazine. Dit artikel is geschreven voor ontwikkelaars die al een goed beeld hebben van Groovy en op zoek zijn naar een creatieve manier om de voordelen van Groovy zinvol binnen Java-projecten te gebruiken.

Alles Groovy

De Groovy community is vele malen kleiner dan de Java-community. Ondanks dat Groovy syntactisch best veel op Java lijkt, is het toch iets heel anders. Groovy code kan dan ook niet zomaar door iedere Java-ontwikkelaar onderhouden worden. Dit is een belangrijke reden om Groovy niet in te zetten voor projecten die lange tijd moeten worden onderhouden.

Voor veel ontwikkelaars houdt de wereld van Groovy hier op. Leuk voor hobbyprojecten thuis, maar simpelweg niet bruikbaar in het echte leven. Door Groovy echter iets subtieler toe te passen op plaatsen waar dit echt meerwaarde biedt, kunnen de voordelen van Groovy benut worden zonder dat dit op grote schaal bovengenoemde problemen met zich meebrengt.

De sterke punten van Groovy

Om een keuze te maken op welke punten Groovy dan het best ingezet kan worden, moeten we eerst analyseren waar deze dynamische taal nu echt iets extra's te bieden heeft ten opzichte van Java. Naar mijn idee is dat het volgende:

- Het runtime toevoegen of wijzigen van gedrag;
- Het builder pattern;
- Eenvoudig gebruik van collections.

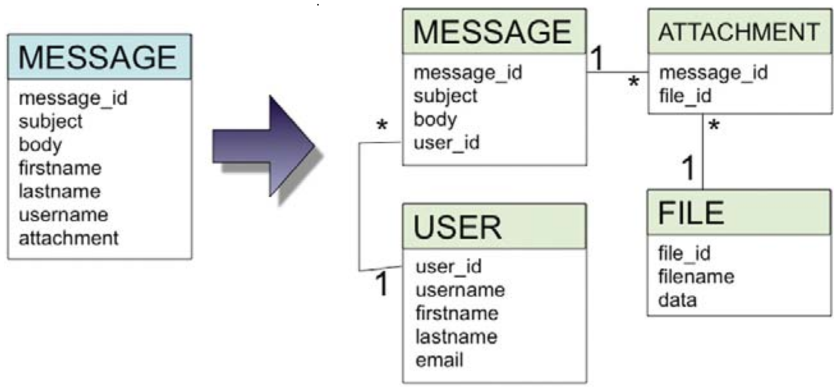
Dat lijkt op het eerste gezicht niet zo heel spannend, maar heeft wel grote gevolgen voor de code die je schrijft. Deze drie punten zijn overigens direct te herleiden naar het dynamische karakter van Groovy. Statische typering is optioneel en methodes kunnen runtime aan een class worden toegevoegd.

Het zijn juist bovengenoemde punten die het mogelijk maken sommige code zoveel eenvoudiger op te schrijven dan in Java. Natuurlijk, vrijwel iedere regel code in Java kan in Groovy op een kortere manier geschreven worden. Dat is in de meeste gevallen echter nog lang niet voldoende winst om tegen de nadelen van het introduceren van een nieuwe taal op te wegen. In bepaalde situaties gaat de winst echter veel verder dan een paar regels minder code schrijven. Op die punten kan Groovy dan een goede toevoeging op de bestaande technologie zijn.

Databasescripts en externe tools

Bij veel projecten worden er scripts of kleine applicaties geschreven voor het uitvoeren van databasegerelateerde taken. Denk hierbij aan het neerzetten

Paul Bakker
is trainer/consultant
bij InfoSupport.



Afbeelding 1: migratie naar een nieuw schema.

Groovy maakt het ontzettend eenvoudig om met een database te werken

van testdata of het migreren van een oud schema naar een nieuw schema. In het tweede voorbeeld wordt er een script gebruikt om data naar een compleet andere structuur om te vormen, zonder referenties tussen tabellen kwijt te raken. Natuurlijk zijn er uitgebreide databasetools om dit te doen, maar dat valt vaak buiten de mogelijkheden van een ontwikkelteam, terwijl een script in veel gevallen voldoet.

Groovy maakt het ontzettend eenvoudig om met een database te werken, veel eenvoudiger dan direct met JDBC. Daarnaast zijn dit soort scripts vaak van tijdelijke aard. Na een eenmalige migratie wordt het script niet meer gebruikt. Dit lost meteen het probleem van onderhoudbaarheid op en daarmee is er geen goede reden meer om Groovy niet voor dit soort taken te gebruiken.
Als voorbeeld is er in codevoorbeeld 1 een stuk van een script opgenomen waarmee de eenvoudige migratie uit afbeelding 1 wordt uitgevoerd.

Het stappenplan voor de migratie is als volgt:

- Maak connectie met beide databases
- Haal alle oude berichten op
- Roep voor iedere rij een 'saveUser' methode aan
- Gebruik het ID van de nieuw opgeslagen 'User' in de 'saveMessage' methode om een referentie tussen een 'Message' en een 'User' te leggen.
- Gebruik het ID van het nieuw opgeslagen 'Message' om een 'saveAttachment' methode aan te roepen.

```
Sql oldDB = Sql.newInstance("jdbc:...")
Sql newDB = Sql.newInstance("jdbc:...")

oldDB.eachRow("select * from messages") {message ->
   def userId = saveUser([firstname: message.firstname,
           lastname: message.lastname, username: message.
username])
      def messageId = saveMessage([subject: message.
subject,
                            body: message.body, user_id:
userId])

   if(message.attachment_data) {
       def fileId = saveFile([data: message.attachment_
data])
      saveAttachment(messageId, fileId)
```

```
    }
}

private int saveUser(def user) {
   def result = newDB.executeInsert(
        "INSERT INTO user (firstname, lastname,
username)
        VALUES ($user.firstname, $user.lastname, $user.
username)")

return result[0][0]
}

//Overige save methoden
```

Codevoorbeeld 1: een gedeelte van een migratie script.

Let er op dat JDBC-drivers een eigen strategie kunnen hebben hoe resultaten ingeladen worden. Bij MySQL worden alle rijen bij een SELECT bijvoorbeeld direct in het geheugen geladen. In veel gevallen is dit het gewenste gedrag, maar voor een migratiescript betekent dit al snel een tekort aan geheugen. Gelukkig is dit gedrag meestal te beïnvloeden. Bij MySQL gebeurt dit in de connectiestring zoals is weergeven in codevoorbeeld 2. Meer details zijn te vinden in de documentatie. Dit is overigens geen Groovy gerelateerd probleem, maar de geheugenproblemen worden zonder deze kennis vaak onterecht bij Groovy gezocht.

```
jdbc:mysql://localhost/olddatabase?defaultFetchSize=500&
useCursorFetch=true
```

Codevoorbeeld 2: een MySQL connectie string met aangepaste fetch size.

Een database migratiescript kan misschien niet door een ontwikkelteam zelf worden uitgevoerd in verband met rechten op de database. Om het script eenvoudig aan een beheerder uit te kunnen leveren kan deze verpakt worden in een executeerbare JAR. Door de Groovy JAR mee te verpakken en op te nemen in de manifest is er geen externe afhankelijkheid met Groovy. Het script is dan als normale Java applicatie op te starten.

Unit testen

Unit testen is vaak pas echt goed te gebruiken als er gebruik wordt gemaakt van mocking. Kort gezegd is mocking het vervangen van aanroepen naar code die buiten de scope van de 'unit' ligt, door aanroepen naar een namaakobject. Het gedrag van dit object wordt geconfigureerd binnen een test. Door het gebruik van mocking is bijna alle code goed te unit testen, ook als er veel afhankelijkheden zijn met allerlei services, databases etcetera. Mocking wordt makkelijk gemaakt door frameworks zoals EasyMock. Een groot nadeel van mocking is echter dat de code per testcase vaak drastisch toeneemt. Dit komt doordat de mock objecten geconfigureerd moeten worden. Dit betekent dat het schrijven, maar ook het onderhouden van tests, een stuk tijdrovender wordt. Groovy kan de unit test code op

twee manieren enorm inkorten. De meest eenvoudige hulp krijg je door dezelfde tests, met behulp van dezelfde mocking frameworks, simpelweg in Groovy te schrijven. Veel van de mock configuratie is namelijk het creëren en vullen van objecten en collecties. Door de veel kortere syntax die Groovy hiervoor biedt wordt de code ook een stuk korter, zoals te zien is in codevoorbeeld 3.

```
productService = createMock(ProductService.class);
def products = [new Product(name: "p1", price: 100),
                new Product(name: "P2", price: 200)]
expect(productService.getProducts()).andReturn(products);
replay(productService);
```

Codevoorbeeld 3: Eenvoudiger gebruik van EasyMock door Groovy maps.

Het gebruik van een mocking framework kan worden vervangen door eenvoudigere Groovy code. Dit heeft wel een consequentie! Bij het gebruik van een mocking framework kan er ook worden gecontroleerd of alle verwachte aanroepen met de juiste parameters zijn gedaan. Dit kan soms de tests een stuk zinvoller maken, maar het creëert ook een hele sterke koppeling tussen de test en de exacte implementatie van de code. Martin Fowler heeft hier een leuk artikel over geschreven. Het nadelige effect hiervan is dat het onderhoud van tests een stuk duurder wordt. Als de implementatie van een methode wijzigt, terwijl het gedrag naar buiten hetzelfde blijft, falen tests toch. Buiten het feit dat dit gerepareerd moet worden, wordt ook de waarde van regressietesten minder omdat het lastiger is te overzien of falende tests op een geïntroduceerde bug wijzen of dat alleen het interne gedrag is gewijzigd. Het kiezen tussen wel of niet sterk koppelen van tests aan de exacte implementatie is een afweging en verschilt per situatie.

Als een eenvoudig mock object, zonder verdere controles, voldoende is kunnen mock implementaties gemaakt worden met gebruik van Expando objecten en maps. Iedere methode die voor de test case door het mock object geïmplementeerd moet worden wordt in de map opgenomen, met een closure als implementatie. Een voorbeeld hiervan is te zien in codevoorbeeld 4.

```
//Creeer een mock implementatie van ProductService
def productService = [
    getProducts: {
          return [new Product(name: "p1", price: 100),
                  new Product(name: "P2", price: 200)]
    }
] as ProductService

//Roep de te testen Java code aan alsof productService een normale implementatie is
Controller controller = new Controller(productService);
```

Codevoorbeeld 4: een mock object m.b.v. de map Expando notatie.

Mocking frameworks hebben helaas ook een groot nadeel. Ze werken alleen goed als dependencies in code met een vorm van dependency injection verkregen worden. Helaas zijn veel bestaande applicaties hier niet op ingericht of maken externe (aangekochte) libraries het vrijwel onmogelijk om goed te kunnen mocken. Een veel gezien voorbeeld hiervan is het gebruik van statische aanroepen naar alle service methoden. Dit is natuurlijk een mooi moment om over refactoring na te denken, maar is niet altijd (realistisch) op grote schaal toe te passen. Groovy kan hier uitkomst bieden. Hiervoor moet wel (op hele kleine schaal) Groovy code in productiecode worden opgenomen.

De oplossing ligt in de mogelijkheden van Groovy om runtime het gedrag van een class te vervangen door nieuw gedrag, ook voor statische methodes. Dat werkt met het metaClass mechanisme zoals te zien is in codevoorbeeld 5.

```
ExpandoMetaClass.enableGlobally()
ProductServiceImpl.metaClass.'static'.getProducts = {->
[new Product(), new Product()]}
```

Codevoorbeeld 5: het vervangen van een statische methode implementatie.

Hier kleeft wel een groot probleem aan. Het metaClass mechanisme werkt alleen voor het vervangen van Groovy methoden! Daarmee is het mechanisme niet direct toepasbaar op Java code. Door het plaatsten van een Groovy proxy tussen de code die getest moet worden en de (statische) service aanroep werkt dit mechanisme echter wel. Het toevoegen van de proxy betekent een kleine refactor slag, maar heeft vele malen minder impact dan het ombouwen naar een DI oplossing. Een voorbeeld hiervan is weergeven in codevoorbeeld 6.

```
//Groovy proxy class
class ProductServiceProxy {
    public static List<Product> getProducts() {
        return ProductServiceImpl.getProducts()
    }
}

//Java statische service
public class ProductServiceImpl  {
    public static List getProducts() {
      throw new RuntimeException("I need a lot of services running...");
}
}

//Aangepaste Java code met service aanroep
List products = ProductServiceProxy.getProducts();

//Code in unit test om implementatie te vervangen
ExpandoMetaClass.enableGlobally()
ProductServiceImpl.metaClass.'static'.getProducts = {->
 [new Product(), new Product()]}
```

Codevoorbeeld 6: een Groovy proxy voor beter testbare code.

Groovy in productiecode

Het wordt natuurlijk nog veel interessanter als delen van productiecode in Groovy geschreven worden. Dan moet er natuurlijk wel goed worden nagedacht welke code in Groovy en welke code gewoon in Java wordt geschreven. Alleen op plekken waar Groovy significante voordelen heeft ten opzichte van Java is het te overwegen om van Java af te stappen. Het gebruik van Groovy is dan te vergelijken met het gebruik van een framework of library. Ook het gebruik van libraries vereist hele specifieke kennis, en code geschreven op basis van een specifieke library is ook niet zomaar door iedere willekeurige ontwikkelaar te onderhouden.

Een voorbeeld hiervan is het produceren van XML. Dit kan in Java worden opgelost door direct een DOM framework te gebruiken of door een framework zoals JAXB of XStream te gebruiken. In beide gevallen is de oplossing waarschijnlijk een stuk complexer dan wanneer de Groovy XML Builder wordt gebruikt. Het stuk code waarin de XML wordt gegenereerd kan dan ook prima in Groovy worden geschreven, terwijl dit wordt aangeroepen vanuit Java code. Een eenvoudig voorbeeld hiervan is weergegeven in code voorbeeld 7. Het gebruik van Groovy is hierbij beperkt gebleven, terwijl wel alle voordelen gebruikt zijn.

```
//Java code die de Groovy methode aanroept
ProductSerializer ser = new ProductSerializer();
String xml = ser.serialize(products);

//Groovy methode voor het genereren van XML
public String serialize(List<Product> products) {
    def sw = new StringWriter()
    def xml = new MarkupBuilder(sw)
    xml.products {
        for (p in products) {
            product {
                name(p.name)
                price(p.price)
                description(p.description)
            }
        }
    }
    return sw.toString()
}
```

Codevoorbeeld 7: groovy vanuit Java gebruiken voor specifieke taken.

Technische onmogelijkheden

Er is bijna niets onmogelijk met het vervangen van Java-code door Groovy-code. Zelfs een EJB 3 Session Bean of een Servlet kan zonder problemen in Groovy worden ontwikkeld en op een standaard applicatieserver wordt gedeployed. Dat moet uiteraard geen reden zijn om te pas en te onpas Groovy in te zetten, maar is wel een belangrijke voorwaarde voor het toepassen van Groovy op plaatsen waar dat zinvol is.

Builden

Een andere vraag is hoe Groovy binnen een Java buildprocess kan worden ingepast. Bij veel projecten is de build gebaseerd op ANT of op Maven. Beide build tools integreren naadloos met Groovy door het gebruik van de joint compiler. Dit is een slim proces die de Groovy en Java compiler integreren, waardoor referenties tussen Groovy- en Javaclasses zonder problemen worden opgelost. De benodigde configuratie is te vinden op de website van GMaven (Groovy plugin voor Maven) en de website van de "groovyc ANT task". Helaas kunnen tools zoals Findbugs en CheckStyle nog niets met Groovy code, dus wordt de code overgeslagen in de rapporten die een build op kan leveren.

Conclusie

Het is zeker aan te raden een goede plugin voor Groovy in de gebruikte IDE te installeren. Zowel Intellij, Eclipse als Netbeans hebben inmiddels goede ondersteuning voor Groovy. «

Referenties

- MySQL configuration properties - http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-configuration-properties.html
- Mocks Aren't Stubs, Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html
- Inversion of Control and the Dependency Injection Pattern, Martin Fowler - http://martinfowler.com/articles/injection.html
- GMaven - http://groovy.codehaus.org/GMaven
- Groovy ANT Task - http://groovy.codehaus.org/The+groovy+Ant+Task

Ben jij dé freelance ICT'er die wij zoeken?

www.it-staffing.nl

Good thinking!