Formation Groovy ET Grails

IHAB ABADI

Sommaire Séance 1

Présentation de Groovy

Groovy VS Java

Rappel des bases du Groovy (Type, chaine de caractères, collections, opérateurs

GroovyBeans

Closures

DSL

Builders

Annotations

Métaprogrammation

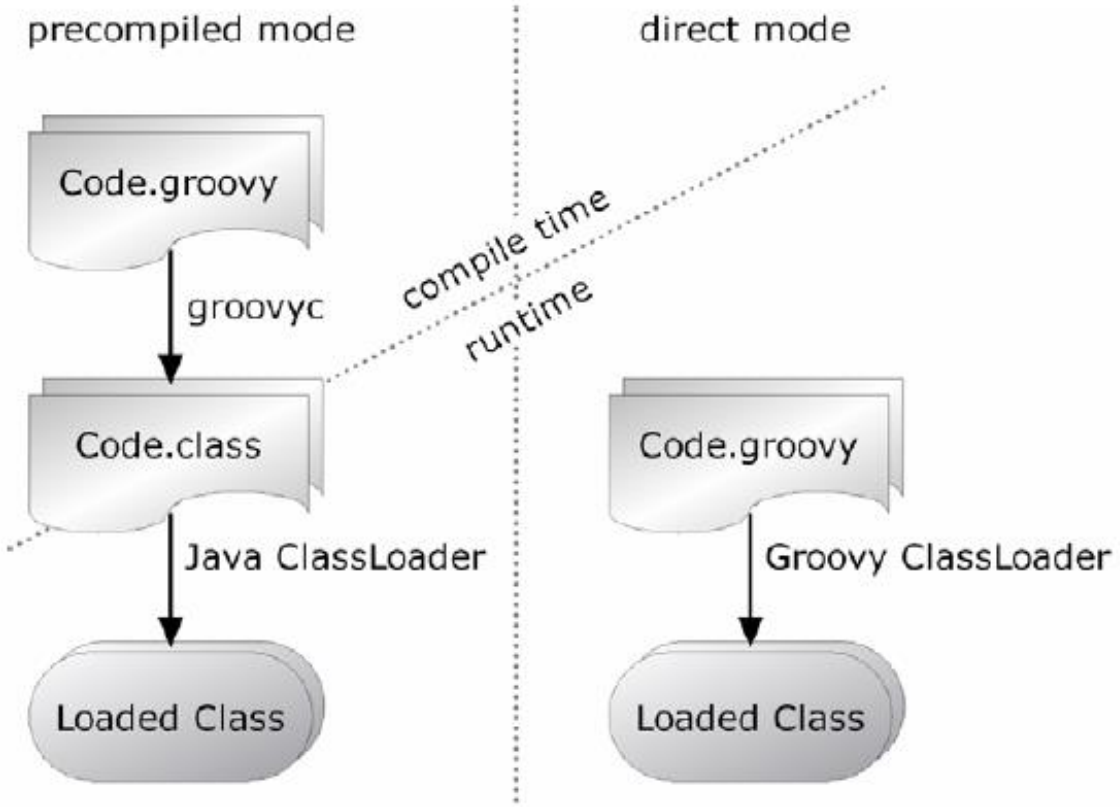
Présentation de Groovy

- Groovy langage orienté objet
- Groovy est un langage dynamique
- Groovy utilise java plateforme
- Groovy compile en byte code
- Groovy utilise la JVM

Présentation de Groovy

- Groovy a une syntaxe très proche de JAVA
- Groovy améliore des projets existants en JAVA
- Groovy utilise compile time mode et runtime mode
- Groovy apporte des fonctionnalités avancées (closures, dynamic typing , meta-programming, …)

Présentation Groovy



Groovy VS Java

- Groovy peut co-exister avec Java
- Groovy utilise JRE
- Groovy est langage plus concis

Groovy VS Java

- Groovy possède un typage optionnel
- Groovy utilise un auto-import de certains packages
- Groovy utilise un mécanisme de multiméthodes
- Groovy et instanciation de tableau
- Groovy et visibilité au niveau du package
- Groovy classe anonyme et imbriquée
- Groovy et chaine de caractère
- Groovy ne possède pas de type primitif

Rappel des bases du Groovy - Type

- Tout est objet en Groovy, les types primitifs sont auto-Wrapper à l'aide d'objet
- Le typage est optionnel en Groovy
- Si type non explicite avec le mot clé def, la variable est considérée comme java.lang.Object
- Le typage en Groovy est safe
- Démo

Rappel des bases du Groovy – Chaine de caractères

- Une chaine de caractère avec simple quotte est considérée comme java.lang.String
- Une chaine de caractère avec double quotte peut être considérée comme java.lang.String ou groovy.lang.GString
- Une chaine de caractère avec triple quotte est considérée groovy.lang.Gstring
- Démo

Rappel des bases du Groovy - Collections

Ranges

Lists

Maps

Rappel des bases du Groovy - Ranges

- Ranges sont des Objets
- Spécifie les limites inférieures et supérieures d'une séquence
- Les limites sont par défaut incluses et peuvent être exclues
- Les ranges supportent des nombres, chaines de caractères, Dates, Ranges inversées
- Ranges possèdent plusieurs méthodes (step, contains, …)
- On peut créer des customs Ranges, en surchargeant les méthodes next et previous et en implémentant l'interface Comparable
- Démo

Rappel des bases du Groovy - Lists

- Les listes en Groovy sont des objets de type java.util.ArrayList
- La syntaxe de déclaration de liste est plus concise avec []
- Les différentes façons de manipuler une liste
- L'utilisation des index négatifs dans une liste
- Les différentes méthodes d'objet List en Groovy
- Les améliorations des fonctionnalités d'une Liste en Groovy à l'aide des opérateurs
- Démo

Rappel des bases du Groovy - Maps

- Les Mpas en Groovy sont des objets de type java.util.HashMap
- La syntaxe de déclaration des Maps est plus concise avec [:]
- Les différentes façons de manipuler une Map
- Les différentes méthodes d'objet Map en Groovy
- Les améliorations des fonctionnalités d'une Map en Groovy à l'aide des opérateurs
- Démo

Rappel des bases du Groovy – quelles que opérateurs spéciales

- Safe opérateur
- Spread opérateur
- Spaceship opérateur
- Mix in en Groovy

GroovyBeans

- Autorise l'accès aux propriétés au JavaBeans définies en Java ou Groovy
- Génération automatique des get / set pour les propriétés des classes Groovy avec une visibilité par défaut
- La génération des get / set se fait uniquement si les accesseurs ne sont pas définis explicitement
- Démo

Rappel des bases du Groovy - Closures

- Les closures sont des objets de types groovy.lang.Closure
- Une fonctionnalité puissante et intégrale de Groovy
- Un bloc de code enveloppé dans un objet.
- Similaire aux classes internes anonymes en Java mais moins verbeuses et plus flexible
- Evite la prolifération des interfaces
- Utilisation des paramètres en closures
- Groovy closures VS expression lambda
- La portée du this des closures, l'utilisation des Owner et delegate
- Les closures sont considéré comme base de la programmation fonctionnelle

DSL

- DSL (Domain Specific Language) est un langage dont les spécifications sont adaptées à un domaine fonctionnel
- Groovy offre des fonctionnalités très avancées pour la réalisation des DSL
- Grails et Gradle utilise les DSL
- Démo de DSL

Exercice

Réaliser le DSL suivant :

```
Email.send {
    from f: "from@from.com"
    to "to@to.com"
    cc "cc@cc.com"
    cc "cc2@cc.com"
    subject s: "subject of email"
    body b: "body of email"
    attach "file1"
    attach "file2"
}
```

Rappel des bases du Groovy – Builders, Slurpers

- Les builders fournissent une surcouche pour la construction d'objets hiérarchiques
- Quelle qu'exemple de builder
- SwingBuilder
- NodeBuilder
- MarkupBuilder
- Les slurpers sont des objets utiles pour la conversion
- Quelle qu'exemple de slurper
- JsonSluprer

Exercice Builder

Réaliser le builder d'une entité compagnie, une compagnie possède une liste de départements, chaque département possède une liste d'employés et chaque employé possède un nom et un role

```
CompanyBuilder builder = new CompanyBuilder()
Company company = builder.company('ABC') {

    department('XYZ') {
        employee('emp12345') {
            name('John')
            role('Administrator')
        }
    }
    department('123') {
        employee('emp987') {
            name('Karen')
            role('Project Manager')
        }
    }
    department('456') {
        employee('emp456') {
            name('Mary')
            role('Developer')
        }
    }
}
```

Annotations

- Les annotations sont une sorte d’interface qui permettent la mise en place des logiques à la fois à la compilation ou exécution
- Les annotations peuvent être utilisées avec tout type d’entités
- Exemple

Exercice

Réaliser une annotation qui permet d’exécuter un comportement différent en fonction des jdk

Métaprogrammation avec Groovy

- Groovy prend en charge deux types de métaprogrammation runtime et compile-time.
- La métaprogrammation runtime permet de modifier le modèle de classe et le comportement d'un programme au moment de l'exécution
- La métaprogrammation compile-time ne se produit qu'au moment de la compilation.

Métaprogrammation runtime

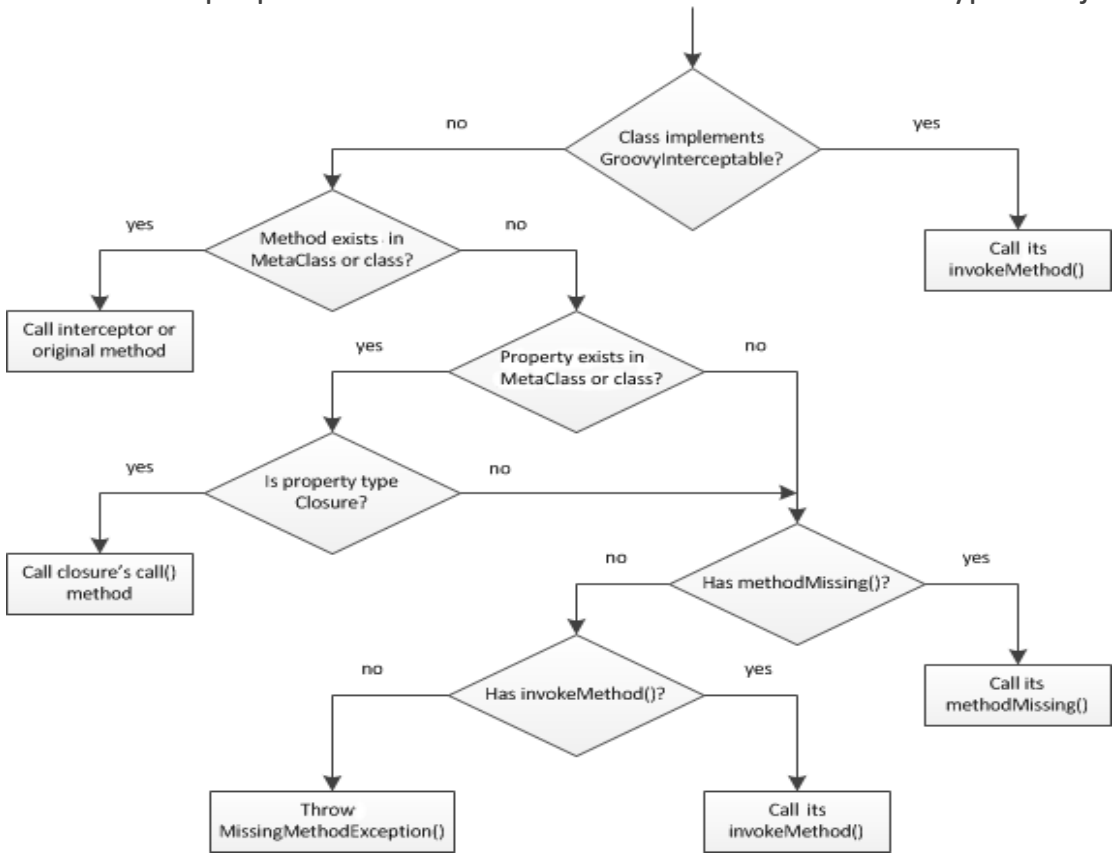
- La métaprogrammation runtime permet de décaler l'exécution en interceptant et injectant des membres de classes et d'interface
- La métaprogrammation runtime se fait en fonction du type d'objet
- Trois type objet en Groovy POJO, POGO, Groovy interceptor

Métaprogrammation runtime

- POJO sont des objets instanciés par des classes java ou tout autre langage compatible avec la JVM
- POGO sont des objets instanciés par des classes écrite en Groovy, ce sont des classes qui héritent de java.lang.Object et qui implémentent l'interface groovy.lang.GroovyObject
- Groovy Interceptor ce sont des objets instanciés par des classes Groovy qui implémentent groovy.lang.GroovyInterceptable

Métaprogrammation runtime

- Invocation des méthodes et propriétés suit différent scénario en fonction du type d'objet



Métaprogrammation compile time

- La métaprogrammation compile time est la possibilité de modifier le code à la compilation
- Ces transformations modifient AST
- La modification se fait au niveau du bytecode
- Avantages du compile-time par rapport au runtime metaprogramming
- Groovy fournit un ensemble de transformation à exécuter à l'aide d'annotation
- Groovy donne la possibilité de créer ces propres AST

Métaprogrammation compile time quelques AST

- AST de transformation de codes : @ToString, @TupleConstructor, @Category, @NullCheck @Builder …
- AST de design : @Singleton, @Mixin, …
- AST Directive de compilation : @Field, @PackageScope,
- Autre AST
- Lien vers Autre AST http://groovy-lang.org/metaprogramming.html

Métaprogrammation compile time –création des AST

- On peut créer deux types de AST
- AST Global
- Cette transformation est appliquée au début de la compilation tout le temps
- Il faut ajouter le chemin vers la classe de transformation dans le service locator META-INF/services/org.codehaus.groovy.transform.ASTTransformation

Métaprogrammation compile time –création des AST

- AST Local :
- Cette transformation est appliquée par annotation
- Il faut créer des annotations qui implémente org.codehaus.groovy.transform.ASTTransformation
- Cette transformation est appliquée uniquement dans la phase d'analyse sémantique de compilation

Utilisation Trait en Groovy

Les traits sont un mécanisme qui permet:

- composition des comportements
- implémentation à l'exécution des interfaces
- Surcharge des comportements

Les traits peuvent être comparé à des interfaces avec des différences majeures.

Démo