GROOVY程序设计与实践

龙辉 2018.5

微博、微信、Twitter：@huihoo

GROOVY是什么?

Apache Groovy

JVM动态、脚本语言、粘合剂

DSL：领域特定语言很重要

Groovy是Gradle和很多CI/CD工具的开发语言和基础

WHY GROOVY

* Groovy最大的优点就是简单，相比Java，它更容易学习和掌握；
* Groovy是Apache基金会项目，有强大的开源社区治理体系；
* Groovy是JVM上的动态、脚本语言；
* Groovy具有Java平台优势，也具有受Python，Ruby和Smalltalk等语言启发的额外强大功能；
* Groovy支持特定领域语言(DSL)和其他紧凑语法，使您的代码变得易于阅读和维护；
* Groovy凭借其强大的原语处理，OO功能和DSL，可轻松编写shell和构建脚本；
* 在开发Web，GUI，数据库或控制台应用程序时，通过减少脚手架代码提高开发人员的生产力；
* 通过支持单元测试和开箱即用模拟来简化测试；
* 与所有现有的Java对象和库无缝集成；
* 直接编译为Java字节码，可以在任何使用Java的地方使用它。

GROOVY可从以下几个方面入手

* 语法（Syntax）
* 语义（Semantics）
* 操作符（Operators）
* 程序结构（Program structure）
* 面向对象 （Object orientation）
* 特征（Traits）
* 闭包（Closures）
* 元编程（Metaprogramming）
* 测试（Testing）
* 设计模式（Design patterns）
* 领域特定语言（Domain-specific language，DSL）

ps: 这也可以作为学习一门新语言的参考路线图。

语法（SYNTAX）

* 注释（Comments）
* 关键字（Keywords）
* 标识符（Identifiers）
* 字符串（Strings）
* 字符 （ Characters）
* 数值（Numbers）
* 数学运算（Math operations）
* 布尔值（Booleans）
* 列表（Lists）
* 数组（Arrays）
* 映射（Maps）

https://github.com/apache/groovy/blob/master/src/spec/doc/core-syntax.adoc

GROOVY关键字

as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	finally
for	goto	if	implements
import	in	instanceof	interface
new	null	package	return
super	switch	this	throw
throws	trait	true	try
while			

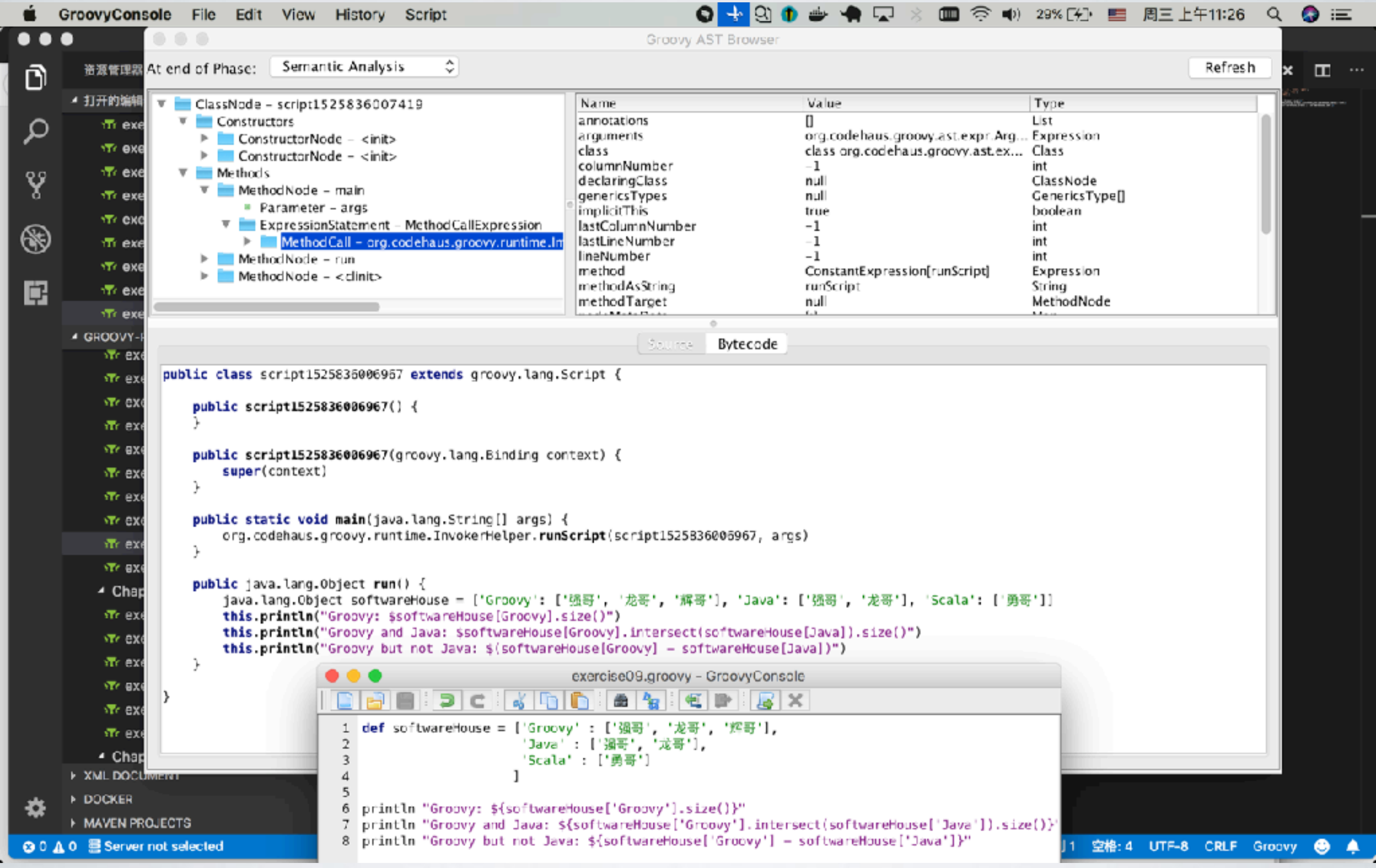
语义（SEMANTICS）

* 语句（ Statements）
—流程控制（Control structures）
—断言（Power assertion）
—标记语句（Labeled statements）
* 表达式 （Expressions）
* 转换/强制（Promotion and coercion）
* 可选择性（Optionality）括号、分号、return、public
* Groovy真相（The Groovy Truth）给出规则判断表达式真假
* 类型（ Typing）// 类型系统是编程语言的核心

思考：Clojure is about Data, Scala is about Types, Java is about Objects.

https://github.com/apache/groovy/blob/master/src/spec/doc/core-semantics.adoc

GROOVY抽象语法树（AST）



操作符（OPERATORS）

* 算术运算符（Arithmetic operators ）
* 关系运算符（Relational operators）
* 逻辑运算符（Logical operators）
* 位操作符（Bitwise operators）
* 条件运算符 （Conditional operators）
* 对象操作符（Object operators）
* 正则表达式操作符（Regular expression operators）
* 其它操作符（Other operators）
* 运算符优先级（Operator precedence）
* 运算符重载（Operator overloading）

https://github.com/apache/groovy/blob/master/src/spec/doc/core-operators.adoc

程序结构（PROGRAM STRUCTURE）

* 包名（Package names）
* 导入（Imports）
* 脚本 vs 类（Scripts versus classes）

https://github.com/apache/groovy/blob/master/src/spec/doc/core-program-structure.adoc

面向对象 （OBJECT ORIENTATION）

* 基本数据类型（Primitive types）
* 类（Class）
* 接口（Interface）
* 构造函数（Constructors）
* 方法 （Methods）
* 字段属性（Fields and properties）
* 注解（Annotation）
* 继承（Inheritance）TBD
* 范型（Generics）TBD

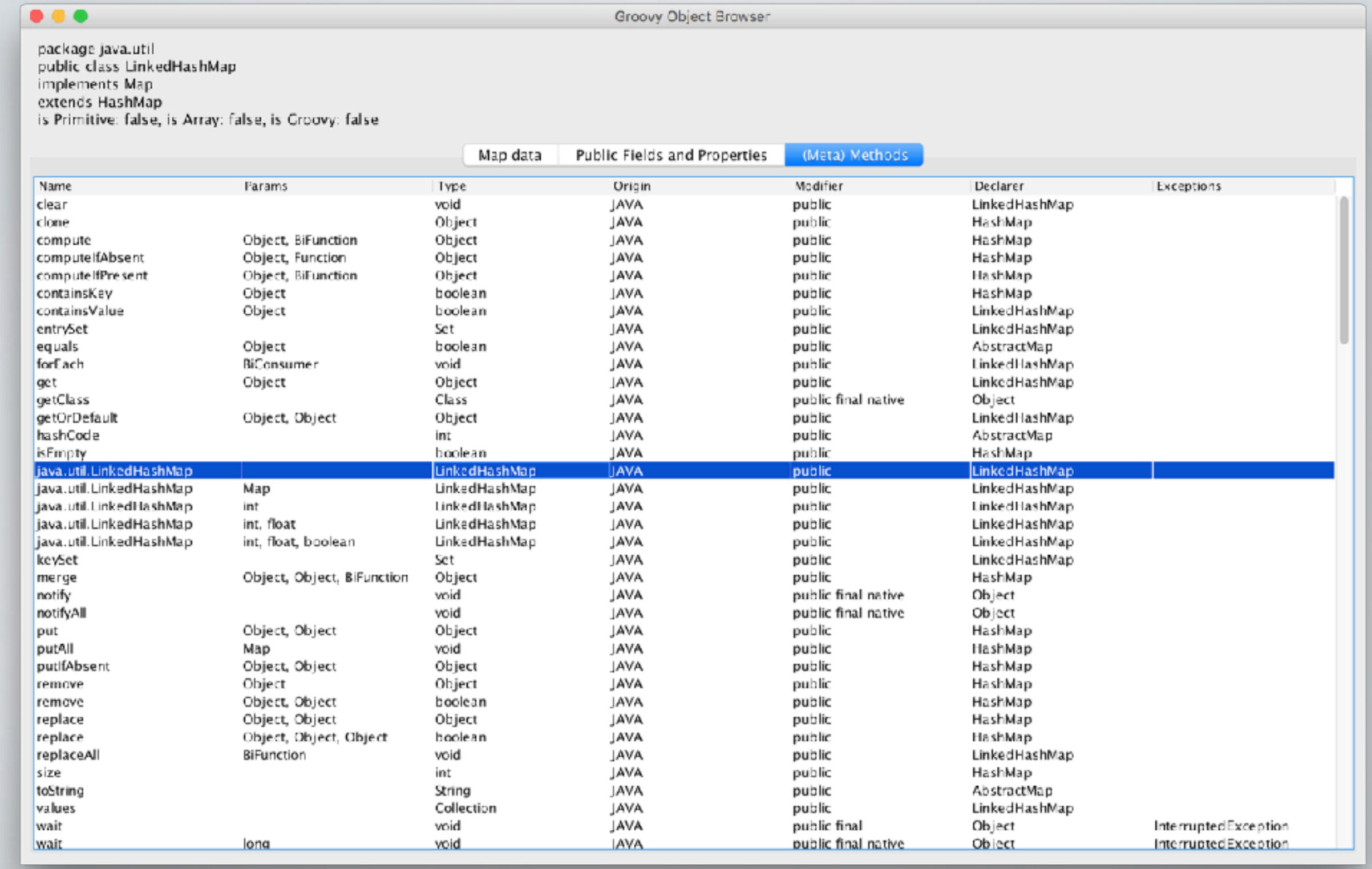
```
class Account {
    def number
    def balance
}

def acc = new Account(number : '123', balance : 1000)
```

Groovy类比Java类代码更少：

* 不需要public修饰符
* 不需要类型说明
* 不需要getter/setter方法
* 不需要构造函数
* 不需要return
* 不需要main

https://github.com/apache/groovy/blob/master/src/spec/doc/core-object-orientation.adoc

对象浏览器


Groovy Object Browser
package java.util
public class LinkedHashMap
implements Map
extends HashMap
is Primitive: false, is Array: false, is Groovy: false
Map data
Public Fields and Properties
(Meta) Methods
Name | Params | Type | Origin | Modifier | Declarer | Exceptions
clear | | void | JAVA | public | LinkedHashMap |
clone | | Object | JAVA | public | HashMap |
compute | Object, BiFunction | Object | JAVA | public | HashMap |
computeIfAbsent | Object, Function | Object | JAVA | public | HashMap |
computeIfPresent | Object, BiFunction | Object | JAVA | public | HashMap |
containsKey | Object | boolean | JAVA | public | HashMap |
containsValue | Object | boolean | JAVA | public | LinkedHashMap |
entrySet | | Set | JAVA | public | LinkedHashMap |
equals | Object | boolean | JAVA | public | AbstractMap |
forEach | BiConsumer | void | JAVA | public | LinkedHashMap |
get | Object | Object | JAVA | public | LinkedHashMap |
getClass | | Class | JAVA | public final native | Object |
getOrDefault | Object, Object | Object | JAVA | public | LinkedHashMap |
hashCode | | int | JAVA | public | AbstractMap |
isEmpty | | boolean | JAVA | public | HashMap |
java.util.LinkedHashMap | | LinkedHashMap | JAVA | public | LinkedHashMap |
java.util.LinkedHashMap | Map | LinkedHashMap | JAVA | public | LinkedHashMap |
java.util.LinkedHashMap | int | LinkedHashMap | JAVA | public | LinkedHashMap |
java.util.LinkedHashMap | int, float | LinkedHashMap | JAVA | public | LinkedHashMap |
java.util.LinkedHashMap | int, float, boolean | LinkedHashMap | JAVA | public | LinkedHashMap |
keySet | | Set | JAVA | public | LinkedHashMap |
merge | Object, Object, BiFunction | Object | JAVA | public | HashMap |
notify | | void | JAVA | public final native | Object |
notifyAll | | void | JAVA | public final native | Object |
put | Object, Object | Object | JAVA | public | HashMap |
putAll | Map | void | JAVA | public | HashMap |
putIfAbsent | Object, Object | Object | JAVA | public | HashMap |
remove | Object | Object | JAVA | public | HashMap |
remove | Object, Object | boolean | JAVA | public | HashMap |
replace | Object, Object | Object | JAVA | public | HashMap |
replace | Object, Object, Object | boolean | JAVA | public | HashMap |
replaceAll | BiFunction | void | JAVA | public | LinkedHashMap |
size | | int | JAVA | public | HashMap |
toString | | String | JAVA | public | AbstractMap |
values | | Collection | JAVA | public | LinkedHashMap |
wait | | void | JAVA | public final | Object | InterruptedException
wait | long | void | JAVA | public final native | Object | InterruptedException

特征（TRAITS）

* 行为组合
* 行为重载
* 接口运行时实现
* 兼容静态类型检测和编译
* 扩展多个特征实现多重继承
* 特征可以扩展另一个特征

```
trait Named {
    public String name
}
class Person implements Named {}
def p = new Person()
p.Named__name = 'Huihoo'
```

思考：特征与接口的区别和联系?

https://github.com/apache/groovy/blob/master/src/spec/doc/core-traits.adoc

闭包（CLOSURES）

定义：Groovy闭包是一种表示可执行代码块的方法。闭包也是对象，可以像方法一样传递参数，闭包常用于处理集合。

Ruby之父松本行弘在《代码的未来》一书中的解释：闭包就是把函数以及变量包起来，使得变量的生存周期延长。闭包和面向对象是一棵树上的两条枝，实现的功能是等价的。

* 语法（Syntax）
—定义闭包（Defining a closure）
—闭包作为对象（Closures as an object）
—调用闭包（Calling a closure）
* 参数（ Parameters）
—普通参数（Normal parameters）
—隐含参数（Implicit parameter）
—可变参数（Varargs）
* 委托策略（Delegation strategy）
—Groovy闭包 vs Lambda表达式（Groovy closures vs lambda expressions）
—所有者/委托/This（Owner, delegate and this）
* GStrings闭包（Closures in GStrings）
* 闭包强制（Closure coercion）
* 函数式编程（Functional programming)

```
class Example {
   static void main(String[] args) {
      def clos = {println "Hello World"};
      clos.call();
   }
}
```

https://github.com/apache/groovy/blob/master/src/spec/doc/core-closures.adoc

元对象编程（METAPROGRAMMING）

元对象编程：用代码编写(生成)代码

一些例子：编译器、解析器、Lex/Yacc、IDL、DSL、JetBrains MPS等等。

了解Groovy对象和Groovy的方法处理能帮助深入理解Groovy的元对象协议（MOP）。

元对象编程或MOP可以用于动态调用方法，并且可以随时创建类和方法。

* 元编程对象类型
— POJO：Java 对象
— POGO：Groovy对象
— Groovy Interceptor：实现了gapi:groovy.lang.GroovyInterceptable接口的Groovy对象

* 运行时元编程（Runtime metaprogramming）

* 编译时元编程（Compile-time metaprogramming）
—可用抽象语法树转换（Available AST transformations）
—开发抽象语法树转换（Developing AST transformations）

https://github.com/apache/groovy/blob/master/src/spec/doc/core-metaprogramming.adoc

GROOVY拦截器


Class implements GroovyInterceptable?
no
yes
Call its invokeMethod()
Method exists in MetaClass or class?
yes
no
Call interceptor or original method
Property exists in MetaClass or class?
yes
no
Is property type Closure?
yes
no
Call closure's call() method
Has methodMissing()?
no
yes
Call its methodMissing()
Has invokeMethod()?
no
yes
Throw MissingMethodException()
Call its invokeMethod()

测试（TESTING）

* 断言（Power Assertions）
* 模拟和存根（Mocking and Stubbing）groovy.mock.interceptor
* GDK方法（GDK Methods）如：combinations、eachCombination
* 工具（Tool Support）
—代码覆盖测试 （Test Code Coverage） Cobertura
—单元测试（JUnit）
—测试套件（TestSuite）GroovyTestSuite用于将所有测试用例封装在一起
—Spock测试（Spock）被用于单元、集成、BDD（行为驱动开发）测试
—Web自动化测试（Geb）

建议：
1、开发即测试，测试也开发；开发测试混合搅拌，不分彼此，即得质量。ps: 开发轮值1个星期测试
2、生成丰富的测试报告集。

http://huihoo.org/jfox/jfoxsoaf/junit-report.html
https://github.com/apache/groovy/blob/master/src/spec/doc/core-testing-guide.adoc

设计模式（DESIGN PATTERNS）

* Abstract Factory Pattern 抽象工厂模式
* Adapter Pattern 适配器模式
* Bouncer Pattern 保镖模式
* Chain of Responsibility Pattern 责任链模式
* Composite Pattern 组合模式
* Decorator Pattern 装饰器模式
* Delegation Pattern 委托模式
* Flyweight Pattern 享元模式
* Iterator Pattern 迭代器模式
* Loan my Resource Pattern 借贷模式
* Null Object Pattern 空对象模式
* Pimp my Library Pattern 图书馆模式
* Proxy Pattern 代理模式
* Singleton Pattern 单例模式
* State Pattern 状态模式
* Strategy Pattern 策略模式
* Template Method Pattern 模版方法模式
* Visitor Pattern 参观者模式

http://groovy-lang.org/design-patterns.html
https://github.com/apache/groovy/blob/master/src/spec/doc/design-pattern-state.adoc … more

模式举例（DECORATOR）

在面向对象设计中，我们应该尽量使用对象组合模式扩展和复用功能。装饰器模式就是基于对象组合的方式，可以很灵活的从对象外部给对象添加所需要的功能，装饰器模式的本质就是动态组合。

举一个跟踪装饰器的例子：

```
class Calc {
    def add(a, b) { a + b }
}

class TracingDecorator {
    private delegate
    TracingDecorator(delegate) {
        this.delegate = delegate
    }
    def invokeMethod(String name, args) {
        println "Calling $name$args"
        def before = System.currentTimeMillis()
        def result = delegate.invokeMethod(name, args)
        println "Got $result in ${System.currentTimeMillis()-before} ms"
        result
    }
}

def tracedCalc = new TracingDecorator(new Calc())
assert 15 == tracedCalc.add(3, 12)
```

https://github.com/apache/groovy/blob/master/src/spec/doc/design-pattern-decorator.adoc

模式举例（ITERATOR & STATE）

举一个迭代模式的例子：

```
def printAll(container) {
    for (item in container) { println item }
}

def numbers = [ 1,2,3,4 ]
def months = [ Mar:31, Apr:30, May:31 ]
def colors = [ java.awt.Color.BLACK, java.awt.Color.WHITE ]
printAll numbers
printAll months
printAll colors
```

再说说状态模式（State Pattern）

因为我们的系统就是由一系列的状态组成：离线、在线、消息已发送、消息已阅读...

说到这，也想聊一下Actor模型，“一切皆是参与者”，与面向对象编程的“一切皆是对象”类似。
参与者模型的特征是，参与者内部或之间进行并行计算，参与者可以动态创建，参与者地址包含在消息中，交互只有通过直接的异步消息通信，不限制消息到达的顺序。
可想象一下系统中的每个进程就是一个Actor，它们都带有邮箱，可与其它Actor通过消息进行交流。
这些消息可以是：Transaction、Event、Heartbeat、Metric、Trace等等。

可考虑引入GPars并发、并行计算框架和调用链监控系统（或APM）

对State Pattern有多种方式和变体来表示：
*面向接口设计
*提取状态模式逻辑
*DSL

领域特定语言（DOMAIN-SPECIFIC LANGUAGE，DSL)

DSL 是一种程序设计语言，不要试图让 DSL 读起来像自然语言。使用 DSL 的最大价值在于领域专家能够读懂，促进交流协作，加速研发速度。如：SQL、HTML

Groovy DSL 包含以下主题和概念：命令链、操作符重载、脚本类、添加成员属性、构造器等等。

* 命令链（Command chains）
* 操作符重载（Operator overloading）
* 脚本类（Script base classes）
* 添加成员属性（Adding properties to numbers）
* 委托给 （@DelegatesTo）
* 编译自定义（Compilation customizers）
* 定制类型检测扩展（Custom type checking extensions）
* 构造器（ Builders）
—MarkupBuilder & StreamingMarkupBuilder
—NodeBuilder
—CliBuilder
—JmxBuilder
—FileTreeBuilder

Groovy支持XML，Groovy能实现BPEL/BPMN。Groovy是BPEL/BPMN DSL的基石，BPEL/BPMN脚本会包含Groovy编译器和运行时。

思考：开发和维护一套社交（移动）电商的标准流程和规则库，并发展电商DSL，形成自己的核心能力。

https://github.com/apache/groovy/blob/master/src/spec/doc/core-domain-specific-languages.adoc
https://github.com/uniba-dsg/betsy

GRADLE构建语言

Gradle Build Language（DSL）

构建脚本也是Groovy脚本，因此也可以包含Groovy脚本中允许的元素，例如方法定义和类定义。

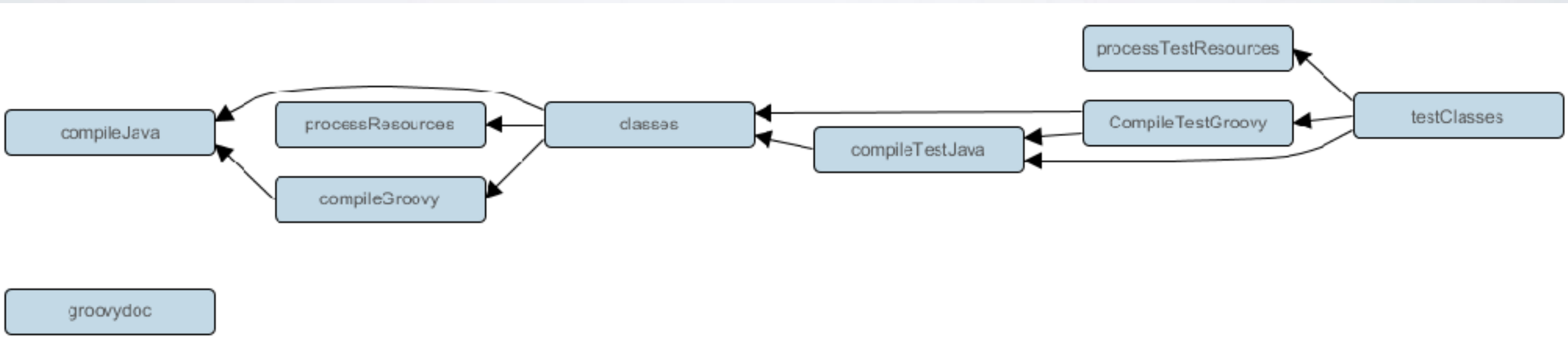
类型：
* Core types
* Container types
* Help Task types
* Task types
* Eclipse/IDEA model types
* Eclipse/IDEA task types
* Native binary core types
* Native binary task types

```
$ touch build.gradle

task hello {
   doLast {
      println 'hello, world'
   }
}

$ gradle -q hello
```

GROOVY插件

该插件支持联合编译，该编译允许您自由组合Groovy和Java代码。一个Groovy类可以扩展一个Java类。

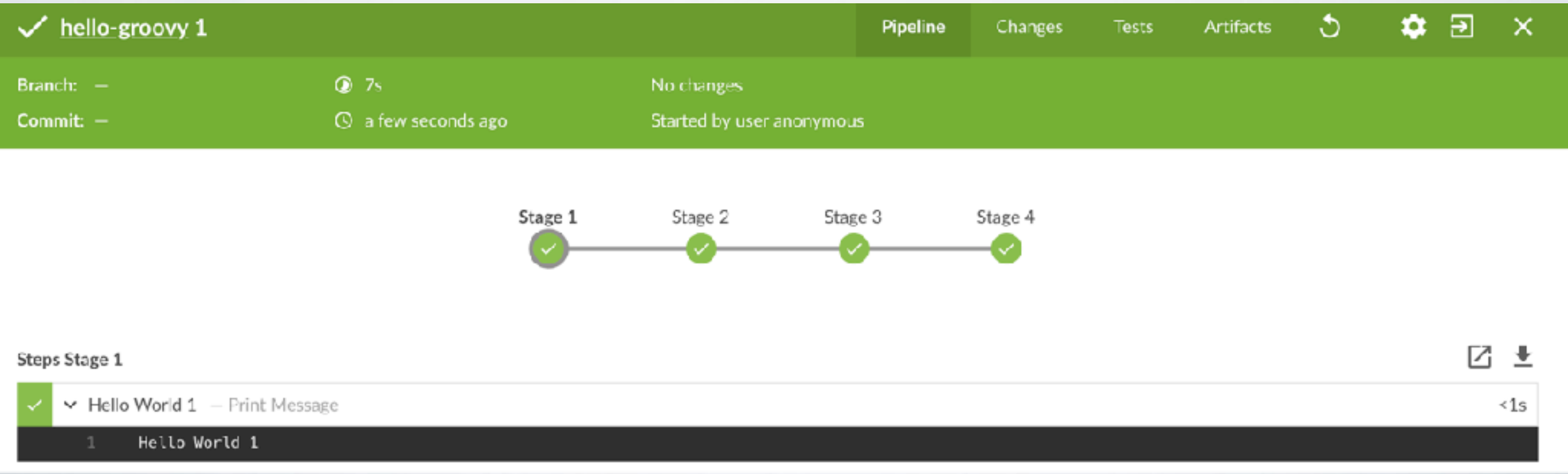
```
$ touch build.gradle

apply plugin: 'groovy'

repositories {
   mavenCentral()
}

dependencies {
   compile ‘org.codehaus.groovy:groovy-all:2.4.12'
   testCompile 'junit:junit:4.12'
}
```



JENKINS流水线

当Jenkins Pipeline首次创建时，Groovy就被选为基础。 Jenkins长期以来一直使用嵌入式Groovy引擎来为管理员和用户提供高级脚本功能。 此外，Jenkins Pipeline的实现者发现Groovy是建立“Scripted Pipeline”DSL的坚实基础。

https://jenkins.io/doc/book/pipeline/



GEB网页自动化测试

Geb基于Groovy和Selenium构建的Web自动化测试框架。

git clone https://github.com/geb/geb-example-gradle
./gradlew chromeTest

过程：
*通过spock定义一系列行为规范：when首页 —> and打开菜单 —> then点击链接 —> when链接被点击 —> then进入页面
*通过junit测试定义的行为规范

https://github.com/geb
https://github.com/SeleniumHQ/selenium
https://www.seleniumhq.org/projects/webdriver/

GROOVY的企业应用

Moqui Ecosystem，Apache OFBiz 的新一代产品和下一代框架。

* Moqui Framework: 核心 Core
* Moqui Mantle: 基础 Foundation
—Universal Data Model (UDM)
—Universal Service Library (USL)
—Universal Business Process Library (UBPL)
* Moqui Crust: 扩展 Add-on

我们可以通过Moqui和Groovy轻松地开发完整的企业级应用

http://moqui.org/

接下来

DSL、自动化、CI/CD、Python

交流问答和在线支持就在知识星球

欢迎大家加入