Java与groovy混编 —— 一种兼顾接口清晰和实现敏捷的开发方式

转载 2018年03月22日 10:55:42

—— 这是我所理解的“工业化开发编程语言”的概念

很显然, java就是种典型的“工业语言”, 非常流行，很多企业靠它赚钱，很实际；
但java也是常年被人黑，光是对其开发效率的诟病就已经足够多，不过java始终屹立不倒；

这样的局面其实无所谓高兴还是担忧，理性的程序员有很多种，其中一种是向“钱”看的 —— 我写java代码，就是因为工作需要而已，能帮助我的组织
目，这很好；
当有人说java语言不好的时候，理性的程序员不会陷入宗教式的语言战争之中，他会思考这些人说的是否有道理；如果真的发现整个java平台大势已去
扭头就走，不过直到目前为止，还没有这种迹象出现;

那么，从这些无数次的口水之争中，我们能否从别人的“战场”上发现一些有用的东西, 来改进我们的开发方式，从而使得java这种已经成为一个“平台”
赚更多的钱呢？
答案是“有的”，感谢那些参与口水战争的、各种阵营的年轻程序员们，有了你们，java speaker们才有了更多的思考;

我就只谈一个最实际的问题：

java被吐槽的这些年, 就开发效率这一点而言，到底有哪些东西是值得借鉴的？

也就是说，到底是哪些主要特性直接导致了某些其它语言在语法上相对于java的优越感？

丰富的literal定义

在groovy中定义map和list的惯用方式：

```
def list = [a, 2 ,3]
def map = [a:0, b:1]
```

而java呢？只能先`new`一个list或map，再一个个add或put进去; 上面这种literal(字面量)形式的写法便捷得多;

而javascript在这方面做得更绝, 我们都用过json，而json其实就是literal形式的object

极端情况下，一门编程语言里的所有数据类型，包括”内建”的和用户自定义的，统统可以写成literal形式;
在这种情形下，其实这种语言连额外的对象序列化、反序列化机制都不需要了 —— 数据的序列化形式就是代码本身, “代码”和“数据”在形式上被统一了

java对这方面几乎没有任何支持，对于提高编码效率来讲，这是值得学习的一点, 起码“内建”数据结构需要literal写法支持

first-class function & higher-order function & function literal(lambda)

无论是js, 还是python/ruby，或是groovy，都可以将函数作为另一个函数的参数传入，以便后者根据执行情况判断是否要调用前者
或者能够将一个函数作为另一个函数的返回值返回，以便后续再对其进行调用
这种高阶函数特性，就不要再说java的匿名内部类“能够”实现了, 如果认为匿名内部类已经”够用”了的话，其实就已经与现在的话题“开发效率”相悖了

高阶函数显然是一种值得借鉴的特性，它会让你少写很多很多无聊的“包装”代码;

还有就是匿名函数(lambda)了
我不喜欢lambda、lambda地称呼这个东西，我更喜欢把它叫做“匿名函数”或者“函数字面量(literal)”, 因为它跟数学上的lambda演算还是有本质区别，
的危险

函数字面量的意思就是说，你可以在任何地方，甚至另一个函数体的调用实参或内部，随时随地地定义另一个新的函数
这种定义函数的形式，除了“这个函数我只想在这里用一次，所以没必要给它起个名字”这种理由之外，还有一个更重要的理由就是“闭包”了

所谓闭包，其实也是一个函数，但是在这个函数被定义时，其内部所出现的所有”自由变量(即未出现在该函数的参数列表中的变量)”已被当前外层上下
xical), 这时候，这个函数拥有的东西不仅仅是一套代码逻辑，还带有被确定下来的、包含那些“自由变量”的一个上下文, 这样这个函数就成为了一个闭

那么闭包这种东西有什么好呢？其实如果懒散而钻牛角尖地想，闭包的所有能力，是严格地小于等于一个普通的java对象的，也就是说，凡是可以用能，就一定可以通过传入一个对象来实现，但反过来却不行 —— 因为闭包只有一套函数逻辑，而对象可以有很多套，其次很多语言实现的闭包其内部象内部属性可变

既然这样，java还要闭包这种东西来干嘛？其实这就又陷入了"匿名内部类可以实现高阶函数"的困境里了 —— 如果我在需要一个闭包的时候，都可以再传入一个对象来实现的话，这根本就跟今天的话题"开发效率"背道而驰了

显然，java是需要闭包的

强大而复杂的静态类型系统

这和开发效率有关么？
编程语言不是越"动态"，开发效率越高么？还需要强大而复杂的静态类型系统么？

试想一下这种api定义：

```
def eat(foo) {
    ...
}
```

这里面你认识的东西可能只有'吃'了, 你知道foo是什么么？你知道它想吃什么么？吃完后要不要产出点什么东西？ —— 你什么都不知道
这种api极易调用出错，这就好比我去买饭，问你想吃什么你说"随便"，但买回肯德基你却说你实际想吃的是麦当劳一样

可能你还会反驳说，不是还有文档么？你把文档写好点不就行了么？ —— 不要逼我再提"匿名内部类"的例子，如果给每个函数写上复杂详尽的文档是然 —— again, 与"开发效率"背道而驰了

那么，静态类型系统，这里显然就该用上了

静态类型系统在多人协作开发、甚至团队、组织间协作开发是非常有意义的；
拥有静态类型系统的编程语言通常都有强大的、带语法提示功能的IDE，这很正常，因为静态类型语言的语法提示功能好做;
只要把别人的库拿过来，导入IDE，各种函数签名只需扫一眼 —— 很多情况下根本不需要仔细看文档 —— 就已经知道这个函数是干嘛用的了, 合作效率

而且，作为"api"，作为"模块边界"，作为与其它程序员合作的"门面", 函数签名上能将参数和返回值类型"卡"得越紧越好 —— 这样别人不用猜你这个函型，甚至他在IDE里一"点"，这里就给自动填上了 :)

要做到"卡得紧"，光有静态类型系统还不够，这个系统还需强大, 试想一下这个例子:

```
/**
 * 我只吃香蕉和猪肉，请勿投食其它物品
 */
public void eat(List<Object> list) {
    for(Object o: list) {
        if(o instanceof Banana){
            ... // eating banana
        } else if(o instanceof Pork) {
            ... // eating pork
        } else {
            throw new RuntimeException("System err.");
        }
    }
}
```

这段纯java代码已经是"定义精确"的静态类型了
但如果没有上面那行注释，你很可能会被`System err.`无数次
而这行注释之所以是必需的，完全是因为我找不到一个比`List<Object>`更好的表达"香蕉或猪肉"的形式, 这种情形足以让人开始想念haskell的either m

在"强大而复杂的类型系统"这一点上，jvm平台上令人瞩目的当属scala了，可惜java没有，这是值得借鉴的

不过这一点的"借鉴"还需java的compiler team发力，我等也只是说说(按照java保守的改进速度，估计`HM`类型系统是指望不上了)

动态类型系统，duck-typing

刚说完静态类型，现在又来说动态类型系统合适么？

然而这与节操无关，我想表达的是，只要是有助于"开发效率"的，都能够借鉴，这是一个理性的java speaker的基本素质

我们在开发项目的时候，大量的编码发生在“函数”或“方法”的内部 —— 这就好比你在屋子里、在家里宅着一样, 是不是应该少一些拘束，多一些直截了
在这种情形下，动态类型系统要不要太爽？ ——

```
Void visitAssert(AssertTree node, Void arg1) {
    def ahooks = this.hooks[VisitAssertHook.class]
    ahooks.each {it.beforeVisitCondition(node, errMsgs, this.ctx, resolveRowAndCol, setError)}
    scan((Tree)node.getCondition(), arg1);
    ahooks.each {it.afterVisitConditionAndBeforeDetail(node, errMsgs, this.ctx, resolveRowAndCol, setError)}
    scan((Tree)node.getDetail(), arg1);
    ahooks.each {it.afterVisitDetail(node, errMsgs, this.ctx, resolveRowAndCol, setError)}
    return null;
}
```

你知道ahooks是什么类型么？你不知道但我(我是编码的人)知道
你知道ahooks身上有些什么方法可以调么？你同样不知道但我知道

你不知道没关系，只要我知道就行了，因为现在是我在写这段代码；
这段代码写完以后，我只会把`Void visitAssert(AssertTree node, Void arg1)`这个类型明确的方法签名提供给你调用，我并不会给你看函数体里面
你知不知道上面这些真的没关系

方法内部满是def, 不用书写繁复的`List<Map<String, List<Map<Banana, Foo>>>>`这种反人类反社会标语, 每个对象我知道它们身上能“点”出些什么来，
来之后`invokedynamic`会为我搞定一切

动态类型系统 —— 这就是方法内部实现应该有的样子
哪怕你的方法内部实现就是一坨shi，你也希望这坨shi能尽可能小只一点，这样看起来更清爽是吧？

不要说我太分裂，我要笑你看不穿 —— 静态类型和动态类型既然都有好处，那么他们能放在一起么？
能的，这里就需要点明这篇文章的政治目的了： “java与groovy混编”
而且，目前来看，jvm平台上，只有它二者的结合，才能完成动态静态混编的任务

曾经我发出过这样一段感叹：

> *公共api、对外接口声明、应用程序边界...这些对外的“脸面”部分代码，如果拥有scala般强大的类型系统...就好了；而私有代码、内部实现、各种*
> *如果拥有groovy般的动态、简单的类型系统...就好了；综上，如果有门语言，在接口和实现层面分别持有上述特性,就好了*

这种“理想”中的语言或许某天我有空了会考虑实现一个

而现在，虽说不是scala，但我终于想要在java和groovy身上来试验一把这种开发方式了
这里我坦白一下为什么没用scala，原因很简单，我在技术选型方面是势利的，scala还不被大多数平均水平的java开发人员(参见”工业化开发编程语言”
这直接导致项目的推进会遇到困难
而相对来讲，我暂且相信大多数java开发人员都还算愿意跨出`groovy`这一小步，当然这还需要时间证明

好了，下面还剩下一点点无关痛痒的牢骚 ——

元编程能力

macro, eval, 编译过程切入, 甚至method missing机制，这些都算“元编程”

元编程能力的强弱直接决定了使用这种语言创作“内部DSL”的能力
java在元编程方面的能力，几乎为0

这是值得借鉴的

与groovy的混编，顺便也能把groovy的元编程也带进来

各种奇巧的语法糖

语法糖，关起门来吃最美味，这也是一种使得“方法内部实现更敏捷”的附加手段
网上随便下载一份groovy的cheat sheet, 都会列举groovy的那些写代码方面的奇技淫巧
这些奇技淫巧，在各种脚本语言之间其实都大同小异, 因为他们本来就是抄来抄去的
结合方法内部的动态类型环境，这一定会进一步缩小方法内部实现代码的体积

java & groovy混编：一种最“势利”的折衷

我不去讨论什么语言才是The True Heir of Java, 那会使这篇文章变成一封战书，我只关心如何更好地利用现有开发资源完成项目，高效地帮组织实现
所以说java和groovy的混编是一种最“势利”的折衷，我不想强迫平均水平的开发人员去学习一种完全不同的语言，短期内不会对项目有任何好处，真正
会找时间去学
而groovy，说它是java++也不为过，因为java代码直接就可以被groovy编译, groovy完全兼容java语法, 对一般java开发人员来说，这真是太亲切了

这里我要提一下我对“java和groovy混编”的一个个人性质的小尝试 —— kan-java项目

kan-java这个小工具，凡是用户在编码使用过程中能“碰”到的类和接口，全部都由java定义, 这确保用户拿到的东西都有精确的类型定义

凡是对上述接口的实现，都以groovy代码的形式存在

这贯彻了”接口静态类型，内部实现动态类型”的宗旨, 或者说“凡是要提供给另外一个人看、调用的地方(接口或接口类)，使用java，否则就用groovy”

当然了，单元测试也完全由groovy代码实现

将kan-java的jar包引入到项目中使用时，就跟使用其它任何纯java实现的jar包一样 —— 接口清晰，参数类型明确，返回类型明确, 你不会也没有必要知
实现的时候，使用动态语言爽过一把

对于java和groovy的混编，项目的pom.xml如何配置，除了可以参考kan-java的配置外，还可以参考这个gist: https://gist.github.com/pfmiles/2f2al
里面举例了两种配置方式，各有特色

具体的效果，还需要真正地去实际项目中体会
另外，kan-java也是一个有趣的工具，这个工具所实现的功能我也是从未见到java世界内有其它地方讨论过的，它可以辅助java做“内部DSL”，有场景

MAR 14TH, 2015 | COMMENTS
—— 这是我所理解的“工业化开发编程语言”的概念

很显然, java就是种典型的“工业语言”, 非常流行，很多企业靠它赚钱，很实际；
但java也是常年被人黑，光是对其开发效率的诟病就已经足够多，不过java始终屹立不倒；

这样的局面其实无所谓高兴还是担忧，理性的程序员有很多种，其中一种是向“钱”看的 —— 我写java代码，就是因为工作需要而已，能帮助我的组织
目，这很好；
当有人说java语言不好的时候，理性的程序员不会陷入宗教式的语言战争之中，他会思考这些人说的是否有道理；如果真的发现整个java平台大势已去
扭头就走，不过直到目前为止，还没有这种迹象出现;

那么，从这些无数次的口水之争中，我们能否从别人的“战场”上发现一些有用的东西, 来改进我们的开发方式，从而使得java这种已经成为一个“平台”
赚更多的钱呢？
答案是“有的”，感谢那些参与口水战争的、各种阵营的年轻程序员们，有了你们，java speaker们才有了更多的思考;

我就只谈一个最实际的问题：

java被吐槽的这些年, 就开发效率这一点而言，到底有哪些东西是值得借鉴的？

也就是说，到底是哪些主要特性直接导致了某些其它语言在语法上相对于java的优越感？

丰富的literal定义

在groovy中定义map和list的惯用方式：

```
def list = [a, 2 ,3]
def map = [a:0, b:1]
```

而java呢？只能先`new`一个list或map，再一个个add或put进去; 上面这种literal(字面量)形式的写法便捷得多;

而javascript在这方面做得更绝, 我们都用过json，而json其实就是literal形式的object

极端情况下，一门编程语言里的所有数据类型，包括”内建”的和用户自定义的，统统可以写成literal形式;
在这种情形下，其实这种语言连额外的对象序列化、反序列化机制都不需要了 —— 数据的序列化形式就是代码本身, “代码”和“数据”在形式上被统一了

java对这方面几乎没有任何支持，对于提高编码效率来讲，这是值得学习的一点, 起码“内建”数据结构需要literal写法支持

first-class function & higher-order function & function literal(lambda)

无论是js, 还是python/ruby，或是groovy，都可以将函数作为另一个函数的参数传入，以便后者根据执行情况判断是否要调用前者
或者能够将一个函数作为另一个函数的返回值返回，以便后续再对其进行调用
这种高阶函数特性，就不要再说java的匿名内部类“能够”实现了, 如果认为匿名内部类已经”够用”了的话，其实就已经与现在的话题“开发效率”相悖了

高阶函数显然是一种值得借鉴的特性，它会让你少写很多很多无聊的“包装”代码;

还有就是匿名函数(lambda)了
我不喜欢lambda、lambda地称呼这个东西，我更喜欢把它叫做“匿名函数”或者“函数字面量(literal)”, 因为它跟数学上的lambda演算还是有本质区别，
的危险

函数字面量的意思就是说，你可以在任何地方，甚至另一个函数体的调用实参或内部，随时随地地定义另一个新的函数
这种定义函数的形式，除了“这个函数我只想在这里用一次，所以没必要给它起个名字”这种理由之外，还有一个更重要的理由就是“闭包”了

所谓闭包，其实也是一个函数，但是在这个函数被定义时，其内部所出现的所有”自由变量(即未出现在该函数的参数列表中的变量)”已被当前外层上下
xical), 这时候，这个函数拥有的东西不仅仅是一套代码逻辑，还带有被确定下来的、包含那些“自由变量”的一个上下文, 这样这个函数就成为了一个闭

那么闭包这种东西有什么好呢？其实如果懒散而钻牛角尖地想，闭包的所有能力，是严格地小于等于一个普通的java对象的，也就是说，凡是可以用
能，就一定可以通过传入一个对象来实现，但反过来却不行 —— 因为闭包只有一套函数逻辑，而对象可以有很多套，其次很多语言实现的闭包其内部
象内部属性可变

既然这样，java还要闭包这种东西来干嘛？其实这就又陷入了”匿名内部类可以实现高阶函数”的困境里了 —— 如果我在需要一个闭包的时候，都可以
再传入一个对象来实现的话，这根本就跟今天的话题“开发效率”背道而驰了

显然，java是需要闭包的

强大而复杂的静态类型系统

这和开发效率有关么？
编程语言不是越“动态”，开发效率越高么？还需要强大而复杂的静态类型系统么？

试想一下这种api定义：

```
def eat(foo) {
    ...
}
```

这里面你认识的东西可能只有’吃’了, 你知道foo是什么么？你知道它想吃什么么？吃完后要不要产出点什么东西？ —— 你什么都不知道
这种api极易调用出错，这就好比我去买饭，问你想吃什么你说“随便”，但买回肯德基你却说你实际想吃的是麦当劳一样

可能你还会反驳说，不是还有文档么？你把文档写好点不就行了么？ —— 不要逼我再提“匿名内部类”的例子，如果给每个函数写上复杂详尽的文档是
然 —— again, 与“开发效率”背道而驰了

那么，静态类型系统，这里显然就该用上了

静态类型系统在多人协作开发、甚至团队、组织间协作开发是非常有意义的;
拥有静态类型系统的编程语言通常都有强大的、带语法提示功能的IDE，这很正常，因为静态类型语言的语法提示功能好做;
只要把别人的库拿过来，导入IDE，各种函数签名只需扫一眼 —— 很多情况下根本不需要仔细看文档 —— 就已经知道这个函数是干嘛用的了, 合作效率

而且，作为”api”，作为“模块边界”，作为与其它程序员合作的“门面”, 函数签名上能将参数和返回值类型“卡”得越紧越好 —— 这样别人不用猜你这个函
型，甚至他在IDE里一“点”，这里就给自动填上了 :)

要做到“卡得紧”，光有静态类型系统还不够，这个系统还需强大, 试想一下这个例子:

```
/**
 * 我只吃香蕉和猪肉，请勿投食其它物品
 */
public void eat(List<Object> list) {
    for(Object o: list) {
        if(o instanceof Banana){
            ... // eating banana
        } else if(o instanceof Pork) {
            ... // eating pork
        } else {
            throw new RuntimeException("System err.");
        }
    }
}
```

这段纯java代码已经是“定义精确”的静态类型了
但如果没有上面那行注释，你很可能会被`System err.`无数次
而这行注释之所以是必需的，完全是因为我找不到一个比`List<Object>`更好的表达“香蕉或猪肉”的形式, 这种情形足以让人开始想念haskell的either m

在“强大而复杂的类型系统”这一点上，jvm平台上令人瞩目的当属scala了，可惜java没有，这是值得借鉴的

不过这一点的“借鉴”还需java的compiler team发力，我等也只是说说(按照java保守的改进速度，估计`HM`类型系统是指望不上了)

动态类型系统，duck-typing

刚说完静态类型，现在又来说动态类型系统合适么？

然而这与节操无关，我想表达的是，只要是有助于“开发效率”的，都能够借鉴，这是一个理性的java speaker的基本素质

我们在开发项目的时候，大量的编码发生在“函数”或“方法”的内部 —— 这就好比你在屋子里、在家里宅着一样, 是不是应该少一些拘束，多一些直截了
在这种情形下，动态类型系统要不要太爽？ ——

```
Void visitAssert(AssertTree node, Void arg1) {
    def ahooks = this.hooks[VisitAssertHook.class]
    ahooks.each {it.beforeVisitCondition(node, errMsgs, this.ctx, resolveRowAndCol, setError)}
    scan((Tree)node.getCondition(), arg1);
    ahooks.each {it.afterVisitConditionAndBeforeDetail(node, errMsgs, this.ctx, resolveRowAndCol, setError)}
    scan((Tree)node.getDetail(), arg1);
    ahooks.each {it.afterVisitDetail(node, errMsgs, this.ctx, resolveRowAndCol, setError)}
    return null;
}
```

你知道ahooks是什么类型么？你不知道但我(我是编码的人)知道
你知道ahooks身上有些什么方法可以调么？你同样不知道但我知道

你不知道没关系，只要我知道就行了，因为现在是我在写这段代码；
这段代码写完以后，我只会把`Void visitAssert(AssertTree node, Void arg1)`这个类型明确的方法签名提供给你调用，我并不会给你看函数体里面
你知不知道上面这些真的没关系

方法内部满是def, 不用书写繁复的`List<Map<String, List<Map<Banana, Foo>>>>`这种反人类反社会标语, 每个对象我知道它们身上能“点”出些什么来，
来之后`invokedynamic`会为我搞定一切

动态类型系统 —— 这就是方法内部实现应该有的样子
哪怕你的方法内部实现就是一坨shi，你也希望这坨shi能尽可能小只一点，这样看起来更清爽是吧？

不要说我太分裂，我要笑你看不穿 —— 静态类型和动态类型既然都有好处，那么他们能放在一起么？
能的，这里就需要点明这篇文章的政治目的了： “java与groovy混编”
而且，目前来看，jvm平台上，只有它二者的结合，才能完成动态静态混编的任务

曾经我发出过这样一段感叹：

> *公共api、对外接口声明、应用程序边界...这些对外的“脸面”部分代码，如果拥有scala般强大的类型系统...就好了；而私有代码、内部实现、各种*
> *如果拥有groovy般的动态、简单的类型系统...就好了；综上，如果有门语言，在接口和实现层面分别持有上述特性,就好了*

这种“理想”中的语言或许某天我有空了会考虑实现一个

而现在，虽说不是scala，但我终于想要在java和groovy身上来试验一把这种开发方式了
这里我坦白一下为什么没用scala，原因很简单，我在技术选型方面是势利的，scala还不被大多数平均水平的java开发人员(参见”工业化开发编程语言”
这直接导致项目的推进会遇到困难
而相对来讲，我暂且相信大多数java开发人员都还算愿意跨出`groovy`这一小步，当然这还需要时间证明

好了，下面还剩下一点点无关痛痒的牢骚 ——

元编程能力

macro, eval, 编译过程切入, 甚至method missing机制，这些都算“元编程”

元编程能力的强弱直接决定了使用这种语言创作“内部DSL”的能力
java在元编程方面的能力，几乎为0

这是值得借鉴的

与groovy的混编，顺便也能把groovy的元编程也带进来

各种奇巧的语法糖

语法糖，关起门来吃最美味，这也是一种使得“方法内部实现更敏捷”的附加手段
网上随便下载一份groovy的cheat sheet, 都会列举groovy的那些写代码方面的奇技淫巧
这些奇技淫巧，在各种脚本语言之间其实都大同小异, 因为他们本来就是抄来抄去的
结合方法内部的动态类型环境，这一定会进一步缩小方法内部实现代码的体积

java & groovy混编：一种最“势利”的折衷

我不去讨论什么语言才是The True Heir of Java, 那会使这篇文章变成一封战书，我只关心如何更好地利用现有开发资源完成项目，高效地帮组织实现
所以说java和groovy的混编是一种最“势利”的折衷，我不想强迫平均水平的开发人员去学习一种完全不同的语言，短期内不会对项目有任何好处，真正
会找时间去学
而groovy，说它是java++也不为过，因为java代码直接就可以被groovy编译, groovy完全兼容java语法, 对一般java开发人员来说，这真是太亲切了

这里我要提一下我对“java和groovy混编”的一个个人性质的小尝试 —— kan-java项目

kan-java这个小工具，凡是用户在编码使用过程中能“碰”到的类和接口，全部都由java定义, 这确保用户拿到的东西都有精确的类型定义

凡是对上述接口的实现，都以groovy代码的形式存在

这贯彻了”接口静态类型，内部实现动态类型”的宗旨, 或者说“凡是要提供给另外一个人看、调用的地方(接口或接口类)，使用java，否则就用groovy”

当然了，单元测试也完全由groovy代码实现

将kan-java的jar包引入到项目中使用时，就跟使用其它任何纯java实现的jar包一样 —— 接口清晰，参数类型明确，返回类型明确, 你不会也没有必要知
实现的时候，使用动态语言爽过一把

对于java和groovy的混编，项目的pom.xml如何配置，除了可以参考kan-java的配置外，还可以参考这个gist: https://gist.github.com/pfmiles/2f2al
里面举例了两种配置方式，各有特色

具体的效果，还需要真正地去实际项目中体会
另外，kan-java也是一个有趣的工具，这个工具所实现的功能我也是从未见到java世界内有其它地方讨论过的，它可以辅助java做“内部DSL”，有场景