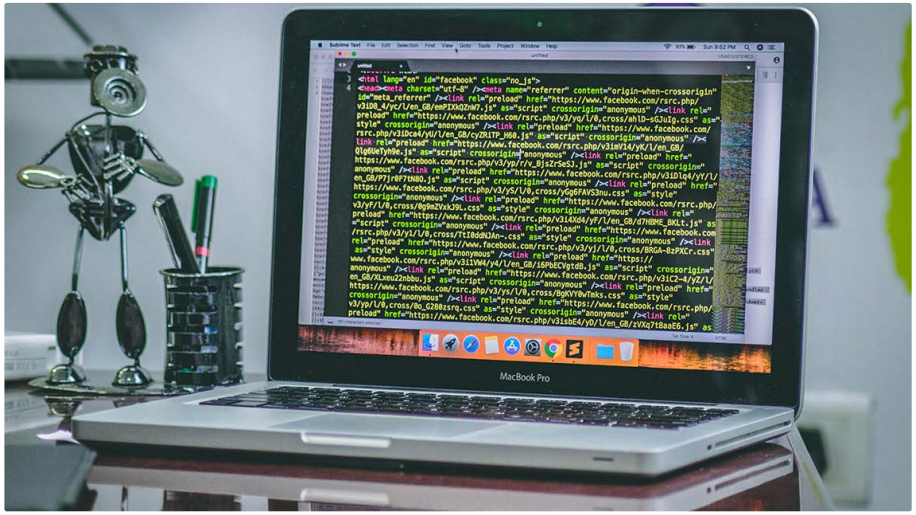
小议Java语言

2018-01-12 朱赟



小议Java语言
朱赟
- 00:00 / 13:22

今天我和你聊聊 Java 语言，这也是我使用最久最熟悉的编程语言之一。

读博士的时候我做过两个领域。刚刚进入莱斯大学的时候进的是程序语言设计组。组里当时有两个教授，分别做两个领域。

一个是瓦利德·塔哈（ Walid Taha ），主要研究领域是类型系统和嵌入式系统的程序语言设计。另一个是罗伯特·卡特赖特（ Robert Cartwright），他喜欢大家叫他 Corky，主要研究的是 Java 的理论和实现。

Corky 教授的研究项目中，有一个叫做 DrJava 的项目。DrJava 是一个轻量级的 Java 开发环境。主要是给在校学生用的，它提供了一个简单直观的交互界面，可以实现类似 Shell 的交互式 Java 代码执行方式。DrJava 在美国大学里有很多人使用，目前已经超过两百万的下载量。

Corky 是个典型的美国教授，与生俱来的花白卷发，清癯的脸庞上一双眼睛炯炯有神，高挺的鼻梁。他常年都是和学生一样的打扮：T恤牛仔裤加运动鞋。出入的时候，背上总是背着一个蓝灰色的电脑包。

当他不谈学术的时候，你会觉得他有着与年龄不相称的单纯和童真。他会和一群学生一起吃系里免费的披萨和点心，也会和其他教授一起站在系里走廊上聊天和哈哈大笑；然而一旦你和他讨论起 Java，他就变得滔滔不绝，整个人散发出特别的魅力。他对 Java 的理解十分深入，我每次和他对话都颇有收益。

虽然我的导师是瓦利德（Walid），但同在一个语言组，平时的研讨班都在一起，我也就有了很多的机会和 Corky 一起讨论各种程序语言的特性和实现。也就是在那个时候，我对 Java 语言有了比较多深层次的了解。

我和 Java 语言的开发者

我的硕士论文是独立实现的一个程序设计语言，包括它的解释编译和用户界面。这个语言主要用于机器人系统的嵌入式编程和仿真，曾经在一家石油公司的井下控制系统开发中被使用。不过因为我导师的离开和种种其他原因，我博士生涯的后三年转了另一个导师做生物信息学的数据分析和建模。

因为有程序设计语言的研究经验，博士毕业找工作的时候，也投了份简历在 Oracle 的 Java 语言开发组。也因为有这样相应的背景，我很顺利地拿到了Java核心类库 （Java Core Library） 开发小组的 Onsite 面试机会。

我去面试的时候应该是 2012 年底，当时面试的那个小组一共好像只有七八个人的样子。Oracle 的面试大部分是白板和聊天，和现在比较主流的面试，上机做题并无 Bug 运行的体验很不相同。我介绍了自己的硕士毕业设计，然后就谈起 Java 新的库或版本可能会增加哪些支持。

2012 年底的时候，Scala 和 Clojure 刚刚火起来不太久，Java 还没有对Lambda 的支持。而当时整个Java团队也正在考虑这件事。话题牵扯到了 Lambda 的实现，正好是我非常熟悉的话题，因为我的导师瓦利德（ Walid ）主要的研究领域就是函数式语言，而对 Lambda 的实现，也是函数式编程语言的核心。

具体的讨论细节我已经不记得了，不过有两点感触颇深：一是他们对于选择哪些函数进核心库（ Core Libarary ）非常谨慎。Java 早期是很轻量级的，后来的版本功能越来越强大，但是语言本身也越来越沉重，这也是很多人喜欢 Scala 的原因。

二是实现函数库的语言开发者对每个函数的精度和运行时间的要求到了令人发指的程度，听说他们有时候读无数的论文，看无数的实现，作大量的比较，就只是为了敲定到底应该在最终的函数中使用哪一种实现方式。

比如浮点数是有舍入误差（ Rounding Error ）的，那么一个数值计算中先算哪一步、后算哪一步带来结果都可能是不同的；而实现中的考虑，往往为了小数点后面十几位以后的一个 1，组里也要反复斟酌很久。

为什么到了 Java 8 才有 Lambda?

很多人抱怨 Java 滞重，语法升级缓慢，过度封装，尤其是对函数式编程支持的不友好等等，其实都和这种谨慎有关。为什么 Lambda 的概念到 Java 8 才有了实现，之前的 Java 版本，包括很多其他语言都没有真正的 Lambda 实现呢？这其实和程序设计语言里的基本概念有关系。

假如我有一个 Lambda 表达式，用伪代码来写，可以写成：

```
def f(x)
    def g()
        return x
    end
    return g
end
```

这个 Lambda 表达式可以看到 f(10) = 10，f(20) = 20。

在一个没有 Lambda 支持，或者嵌套式函数定义支持的语言中 —— 比如 C 语言，这个可能会实现成：

```
typedef int (*fp_t)() ;

int g () {
    return x ;
}

fp_t f(int x) {
    return g ;
}
```

但是问题就在于，g 函数中的 x 是没有定义的，程序不可能编译运行。解决这个问题，我们可以引入一个全局的 x 变量，在对函数 f 进行定义的时候，给这个全局 x 赋值。但是由于 C语言不能每次运行时定义一个新的函数，因此，如果赋值：

```
a = f(10)
b = f(20)
```

那么，虽然我们希望得到 a=10，b=20，但是上面的实现只能给我们 a=20，b=20。

由此可以看出，仅仅的一个匿名函数（ Anonymous Function）， 或者函数指针，是不足以正确地实现 Lambda 的；而正确实现 Lambda，或者说允许把 Lambda 表示的函数作为一个像其他类型的值一样作为参数来传递，语言必须要有对 Lambda 的函数表达，以及一个用来在各层中传递参数值的“参数定义环境”两者同时的实现。这也就是函数语言中的闭包的概念。

换句话说，实现 Lambda 可以作为一个普通类型一样的值来存储和传递，我们需要一个闭包（ Closure ），而闭包（ Closure ）可以看成：

闭包= Lambda 表达式 +记录所有函数局部变量值在每一层 Lambda 中的赋值的一个环境。

实现闭包大体有两种方式。一种叫做“自底向上”的闭包转变，也称为 Flat Closure。它从函数定义的最里层，将每一层的局部函数变量值拷贝到次里层。每一层的变量可能重名，而这就需要变量名解析的技术，对变量按层重命名。这样逐层拷贝，最后形成一个 Lambda 对应的单层的变量赋值环境。

另一种叫做“自顶向下”的闭包转变，也称为共享闭包（ Shared Closure）。它从函数定义的最外层，将每一层的局部函数变量赋值用类似指针的方式传播共享到里层的 Lambda。这种实现的好处是避免重命名和拷贝，但是实现赋值环境的共享其实是很棘手的。

总而言之，Lambda 在语言中的实现是复杂并且昂贵的。不仅容易出错，还会给语言的垃圾收集（GC）带来新的挑战。它也让语言的类型系统的所有证明和推导变得复杂无比。虽然现在主流的语言都提供了Lambda 的实现，但用起来还是有一定限制也需要一些谨慎的。

比如， C 语言仍然不支持嵌套式的函数定义。C++11 增加了对闭包的支持，但是因为语言本身没有垃圾收集的原因，使用起来需要异常谨慎，很容易引起悬挂引用（Dangling References）。

比如，Ruby函数不能直接作为参数传递，而是通过 Method 或者 Proc 来使用。且函数的嵌套定义并没有很好的对作用域进行嵌套；而 Java 8，虽然有了对Lambda 的支持，但是 Java 类型系统（ Type System ）并没有对函数类型有任何的支持。换句话说，Java 8 中其实并没有对函数类型的类型系统的实现，这就意味着一些 Lambda 相关的类型错误，在编译时间可能无法被发现。

看完了这些你就会知道，一门编程语言的变革是多么艰难和复杂。好在 Java 9 已经发布了，Java 语言有了更高和更新的起点。

Java 开发中的常见问题

很多新人入门会要求我推荐编程语言，Java 属于我推荐的语言之一，因为 Java 标准、规范，是面向对象编程的代表，Class、Object、Interface、Abstract、Public、Private、Override 等关键词显式清晰，一旦使用就不会混淆，在学习其他编程语言的时候还可以参考互通。

另外，由于 Java 的流行和开放性，围绕 Java 语言形成了最为广泛的开发平台，不仅有 Spring 这样开源生态社区，在 Java 平台之上还衍生出了很多轻量级的编程语言，比如 Scala、Groovy、Clojure 、Kotlin，这些语言都可以运行在 JVM 之上，形成了非常有生命力的生态环境。

那么在用 Java 开发的过程中需要注意哪些问题呢？

这里的问题并不是指哪些常见的“Java编程最常犯的错误”，比如：

1. 如何正确地将数组转化为列表；
2. 如何判断一个数组是否包含一个值；
3. 如何在一个循环中删除一个列表中的元素；
4. 什么时候用哈希表（ HashTable），什么时候用哈希映射（ HashMap）；
5. 不要在集合中使用原始类型；
6. 如何正确设定访问级别；
7. ArrayList 与 LinkedList 的对比；
8. 可变与不可变；
9. JUnit 中@Before 和@After行为的顺序；
10. SPI 中参数如何设定；
11. API 不要返回空值。

这里想说的是，从架构和规范的角度，如果你选择了使用 Java 进行开发后，应该注意的那些问题。

1 不要重复造轮子，也不要搬太多不同型号的轮子来用

Java 语言有非常成熟的技术社区，在 Java 的生态里有很多相关的库、插件和工具，大部分是成熟技术。比如DW（ Dropwizard），基于 Java 生态中稳定的库构建了一个轻量级的框架，可以支持系统中的配置、日志、序列化等操作，还能构建基于 RESTful 的微服务架构。

如果一个公司内有不同背景的 Java 工程师，有时他们会根据之前公司的工作经验，选取自己熟悉的库和工具，这时就有可能会引起系统中（比如序列化）使用了两种“轮子”。

如何确保大家使用统一的库和工具呢？有很多途径，比如指定编程规划，或是使用工具完成某个服务的初始化，自动配置公司常用的库，或者通过统一的代码审核，让一些资深程序员把关等等。

2 深入了解依赖注入，选择一个好框架

稍具规模的 Java 代码库，都会有复杂的类和对象之间的相关性和依赖性。确保所有工程师理解并使用一个统一的依赖注入框架（比如 Guice）会让很多代码的风格保持一致，避免类的初始化过于复杂。

(注：Guice 是谷歌推出的一个轻量级依赖注入框架，帮助开发者解决 Java 项目中的依赖注入问题。如果你使用过 Spring 的话，会了解到依赖注入是个非常方便的功能。不过，假如你只想在项目中使用依赖注入，那么引入 Spring 就显得太重了，这时候可以考虑采用 Guice。)

3 所有项目使用统一的文件夹结构

在构建项目的时候，工程师们争论最多的问题之一，就是代码文件怎么组织。库放哪里，工具类Utils 文件夹放哪里，哪些做成单独的 Service文件夹，哪些是 Client文件夹，分别又有哪些类。这个需要早期的工程师或者资深工程师统一协调，保证代码库的整体结构自顶而下都结构清晰，不同项目里的文件结构也尽可能遵循同样的组织方式。

（注：utils、service 和 client 都是常用的代码文件夹名称。）

4 规范所有项目的 API，保持统一的风格和模式

API 的接口定义是不是使用 了 IDL（Interface Description Language），服务器（Service） 之间的交互协议是不是一致的，比如基于 HTTPS，内部功能模块和数据 Schema 之间的分层和转化是否有统一的标准。如果在开发过程中，不同服务器或者模块采取了不同的接口模式，让集成变得异常复杂。

5 规范开发环境，提供合适的工具

可能每个人都经历过这样的场景，用自己的编辑器打开代码库的代码之后，做一下自动格式化，或者新增一些代码，结果缩进、空行、括号等等，突然都不一样了。在协同开发的时候，经常会出现多人修改同一批代码的状况，如果频繁遇到代码不规范会非常麻烦，并且会出现很多无效格式的提交（ Commit ）。

要保证所有人的代码格式设置一致，才能避免这样的问题。定义一些标准的格式文件，做成编辑器插件，要求每个开发者使用统一的格式定义，就能避免上述的问题。

总结一下，今天我和你谈了我上学时是如何接触和使用 Java 语言的，讲述了 Java 语言开发者的故事，探讨了为什么 Java 很晚才支持 Lambda 这样的语言特性，最后介绍了在实际开发过程中应该注意的一些问题，希望对你有帮助。

如果你还不知道应该学习哪一门编程语言，那么就从 Java 开始吧。感谢你的收听，我们下期再见。

bluze

2018-01-12

Java看了两周，找了一份工作，以战养兵。期待安妞开设Java系列的技术高阶课程！

作者回复

2018-01-13

我去劝劝，看是否有第二季

Silence

2018-01-12

每过段时间看自己以前写的代码都有一种，“我去，这代码哪个傻瓜写的，肯定不是我”的感觉

有福

2018-01-15

为了这个文章直接订阅了专栏

Geek_746d06

2018-05-07

虽然我是php开发，但是我还是认真的读完就！

zhuhf

2018-01-13

工具文件夹可能叫util比较合适，个人偏见。

闫飞

2018-01-13

第一个例子好像有点不对啊，f返回的是一个匿名函数，所以f(10)得到的是个函数对象，必需要再加一层括号才能求值，得到调用值。

Java用接口的概念和多态来封装函数对象，和C++某些方面有异曲同工之处，但是其stream api和optional类型提供了极好的monad实现。

c++的类型系统对函数式支持更加严格一些，尤其是fp和自动类型推断以及泛型元编程的支持水乳交融，非常巧妙，不像Java尝试用OO来搞定一切。

王建Tyrion

2018-01-12

上个月刚刚把代码重构一番，按照标准的文件夹规范，上个版本真是自己能把自己看晕(=_=)

iLeGeND

2018-05-21

为什么留言不显示呢

怎么肥四

2018-01-13

吉利