链滴

JVM 动态语言支持详解

作者：zwxbest

原文链接：https://ld246.com/article/1544528638983

来源网站：链滴

JDK 7 增加了对 JSR 292 的支持，在 JVM 中动态类型语言的运行速度将变得更快。这一支持的关键于增加了新的 Java 字节码，invokedynamic，它用于方法调用，还有新的连接机制，其中包含了一新的构造：方法句柄（method handle）。

动态类型语言和 JVM

JVM 可以执行 Java 程序，将其编译机器独立的字节码。事实上，任何可以使用有效 class 文件表述功能性语言，都可以运行在 JVM 上。

多年来，运作在 JVM 上语言一直在增加，从 Armed Bear for Common Lisp 到Yoix。动态语言的 JM 实现也越来越多，比如 JRuby 和 Jython，以及 Groovy

动态语言的灵活性，尤其是脚本语言，对于实验性、原型应用程序以及需频繁更新的程序，都具有独的吸引力。这种灵活性源自动态类型。动态类型语言中运行时（runtime）验证程序中的值是否与预类型一致，相对的，静态类型语言，如 Java，是在编译期间检查变量类型，而不是值类型。值得一提是，Java 平台上另一个前景很被看好的静态语言就Scala,包括Java之父和Groovy创始人在内的很多发者都很看好Scala这个强类型的、可扩展性良好的静态语言。

通常，动态类型比静态类型更具灵活性，因为前者允许程序根据运行时的数据生成类型。不过静态类语言的执行更为高效，因为它能够在编译期间排除错误。

动态类型固有的灵活性与 JVM 的执行效率，合二为一。很明显，这就是它能够吸引动态编程语言创者以及使用这些语言构建应用程序的开发者的原因。

JSR 223 动态语言支持的第一步

JSR 223: Scripting for the Java Platform 是将动态语言引入 JVM 的第一步，它是一个规范，定义从动态脚本语言代码访问 Java 代码的 API 接口。它还指定了一个 framework 框架，用户在 Java 应程序中运行脚本引擎。该规范及其实现使得包含 Java 和脚本代码的应用程序的创建更为容易。

动态类型语言的问题

为运行在 JVM 上的动态类型语言开发引擎，必须满足 JVM 所执行的 Java 字节码的要求，而字节码为静态类型语言设计。对于引擎开发者，当生成字节码用于方法调用，这种设计一直都是棘手的难点。

方法调用的字节码要求

静态类型语言中编译时进行类型检查，意味着方法调用，以及它生成的字节码，需要知道该方法返回值类型，以及调用中指定的参数类型。

下面为一段 Java 代码：

```
1.  String s = "Hello World";
2.  System.out.println(s);
```

这里参数类型是已知的。System.out.println()并不返回值，如果方法返回值，需要指定返回值的类。

以上代码相应的字节码如下：

```
1.  ldc #2
2.  astore_1
```

```
3.  getstatic #3
4.  aload_1 invokevirtual #4 // Method java/io/PrintStream.println:(I)V
```

JVM 中字节码的执行通常包含对操作对象栈（operand stack）中值的操作。操作栈是一个相当于硬寄存器的虚拟机。通常，字节码会指示 JVM 局部值压入操作对象栈，将值从栈中取出放进局部变量，复制或交换栈中的值，或者执行生成或使用值的操作。

请看 invokevirtual 一行，它调用了一个方法，而不是对操作对象栈进行操作。从该行注释，我们可看到，它指出了以下信息：

- 提供方法的接收器（receiver）类：java.io.PrintStream
- 方法名称：println
- 方法参数类型：(I) 表示 Integer
- 方法返回值： V 表示 void

这些信息相当于方法的签名。JVM 查找具有该签名的方法，在这里，就是 java.io.PrintStream 类中的 println:(I)V。如果该方法不在那个类中，JVM 将在类的子类中继续查找。

满足要求所进行的拙劣尝试

为了让动态类型语言满足字节码对方法调用的要求，已经有了多种尝试，但没有一种是理想的。

以下面的代码为例：

```
1.  function max (x,y) {
2.    if x.lessThan(y) then y  else x
3.  }
```

接收器和参数都没有指定类型，而对于动态类型语言，直到运行时才提供类型信息，因此，以上代码能满足方法调用需提前获悉类型的要求，也就不能在 Java 平台上成功地编译为字节码。

问题的解决方法之一是为返回值和方法参数创建虚假的（synthetic）Java 类型。在这里，虚假表示真实存在。例如，动态类型语言在实现是可能将代码更改为：

```
1.  Interface50 function max (Interface 51 x,Interface52 y) {
2.   **if** x.lessThan(y) then y **else** x
3.  }
```

类型 Interface 51 和 Interface52 并不存在，只是为了满足相应的要求而指定。

另一种方法成为映射调用（relfected invocation），使用 java.lang.reflect.Method 对象调用方法而避开直接调用方法。这样也就绕开了指定类型的要求。

第三种方法是为动态语言的实现创建一个独特的方法调用解释器（interpreter），以运行在 JVM 上。

虚假类型满足了 Java 字节码的要求。但这种方法不但繁复而且会带来问题。如果动态语言引擎需要改，相应的实现器（implementer）必须重新创建虚假 Java 类型，这种操作常会出错。

运行调用也有其自身的局限。例如，java.lang.reflect.Method 对象提供了动态语言所需的方法访问但对象必须是运行时可用的特定 Java 类型。虽然，动态语言可以在运行期间提供类型信息，但不是可以通过用户映射的规范 Java 类型。

JSR 292 —— 动态语言支持的下一步

JSR 292为 JVM 引入了一个新的 Java 字节码指令，invokedynamic，以及一个新的方法连接机制。

方法调用的字节码指令

Java 虚拟机规范指定了 4 个字节码，用于方法调用：

- invokevirtual
- invokeinterface
- invokestatic
- invokespecial

新的 invokedynamic 指令

新的 invokedynamic 字节码指令的语法与 invokeinterface 指令类似：

1. invokedynamic

但，它的 < method-specification> 只需指定方法名称，对描述符的唯一要求是它应引用非空对象。

invokeinterface 字节码指令差不多是这样的：

1. invokedynamic #10;
2. //DynamicMethod java/lang/Object.lessThan:(Ljava/lang/Object;)

重要的是，invokedynamic 字节码指令运行动态语言的实现器（implementer）将方法调用编译为节码，而不必指定目标的类型，该目标包含了方法、调用的返回类型或方法参数类型。这些类型对于行指令的 JVM 不必是已知的。但如果未提供接收器的类型，JVM 如何找到该方法？毕竟，JVM 需要接并调用真实类型上的真实方法。答案在于，JSR 292 还包含了一个新的动态类型语言的连接机制。JM 使用新的连接机制获取所需的方法。

新的动态连接机制：方法句柄（method handle）

JDK 7 包含了新包，java.dyn，其中包含了与在 Java 平台中动态语言支持相关的类。其中一个类为 MthodHandle。方法句柄是类型 java.dyn.MethodHandle 的一个简单对象，该对象包含一个 JVM 方的匿名引用。

新连接机制还包含一个引导方法（bootstrap 方法），它是一个方法句柄，决定了调用现场（call sit）调用的目标方法。调用现场是调用指令的实例，在本节中，它就是 invokedynamic 字节码指令的例。包含 invokedynamic 指令的每个类都必须指定引导方法。

JVM 第一次遇到具有接收器和参数的 invokedynamic 字节码时，它调用引导方法。调用语言支持的法，可以使用术语 up-call 来描述。

引导方法反过来选择相应的目标方法句柄。然后 JVM 将该方法句柄引用的方法与 invokedynamic 节码关联起来。JVM 下次遇到具有相同接收器和参数的 invokedynamic 字节码时，它将立即调用之所选的方法。

方法句柄相当简单，仅包含一个描述特定类型的类型令牌（type toke）。此外，方法句柄隐式地包一个与其关联的 invoke 方法。要调用方法句柄，你需要调用它的 invoke 方法，与调用对象方法类，即 MethodHandle.invoke(...)。由于每个方法句柄都具有其自身的类型，因此，它只接受那个类型 invoke 调用。如果调用的类型与方法句柄的类型不匹配，方法句柄将返回异常。

总之，方法句柄提供了一种连接机制，它能够让 JVM 根据 invokedynamic 字节码指令调用正确的法。但 JVM 遇到 invokedynamic 字节码时，它将使用方法句柄获得所需的方法。请注意，相对于射调用，方法句柄提供了一种更好的方式，来满足方法调用的字节码要求。相较而言，方法句柄提供一种命名和连接方法的方式，而无需考虑方法类型或位置，而且这种方式具有完善的类型安全和本地执行速度。

通过接口注入（interface injection）在运行时修改类

接口注入能够在运行时修改类，这样类就可以构建新的接口。对于动态类型语言，尤其是基本语言，是一个常见的功能。但它不属于 JVM 标准的一部分。该功能还处于调研阶段，以便加入 JSR 292 中。

在 JVM 中支持接口注入，运行时语言将可以推荐新的功能，以模块化的方式供其自身使用。例如，设 JVM 在运行的语言的类或类集合需要串行化的类型，而它尚未在该语言实现。运行时该语言可以义一个串行定义为可注入的接口。它还可以定义一个注入方法。该方法定义该语言将为哪个类指定新串行能力。对相关对象调用该注入方法，就可以完成注入。利用接口注入，可以使 JVM 中的动态类语言很方便地与 JVM 中其他语言进行整合。

总结

多年来，在 JVM 上运行的语言越来越多。在 JVM 中支持动态类型语言，对于使用动态语言的开发者常具有吸引力。因为，动态类型让开发者更具灵活性，而且 JVM 具有更好的执行效率。但是，对于态类型语言，满足方法调用的字节码的要求非常困难。为了应对这一难题，JSR 292 提供了新的字节码 invokedynamic 以及新的基于方法句柄的连接机制。此外，目前还在进行调研在 JSR 292 中引入接注入，它能够在运行修改类，从而可以实现新的接口。