|23.1 Interpreter模式

【语法规则也是类】

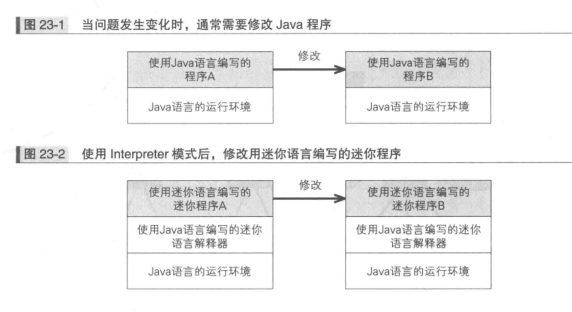
学习到这里，大家应该已经掌握了不少设计模式。设计模式的目A的之一就是提高类的可复用性。可复用性是指不用做太大修改（甚至是不做任何修改）就可以在多种应用场景使用之前编写的类。

在本章中，我们将学习Interpreter模式。

在lnterpreter模式中，程序要解决的问题会被用非常简单的“迷你语言”表述出来，即用“迷你语言”编写的“迷你程序”把具体的问题表述出来。迷你程序是无法单独工作的，我们还需要用Java语言编写一个负责“翻译”（interpreter）的程序。翻译程序会理解迷你语言，并解释和运行迷你程序。这段翻译程序也被称为解释器。这样，当需要解决的问题发生变化时，不需要修改Java语言程序，只需要修改迷你语言程序即可应对。

下面，我们用图示展示一下当问题发生变化时，需要哪个级别的代码。使用Java语言编程时，需要修改的代码如图23-1所示。虽然我们希望需要修改的代码尽量少，但是多多少少都必须修改Java代码。

但是，在使用Interpreter模式后，我们就无需修改Java程序，只需修改用迷你语言编写的迷你程序即可（图23-2）。



|23.2迷你语言

|迷你语言的命令

在开始学习Interpreter模式的示例程序之前，我们先来了解一下本章中涉及的“迷你语言”。迷你语言的用途是控制无线玩具车。虽说是控制无线玩具车，其实能做的事情不过以下3种。

·前进1米（go）

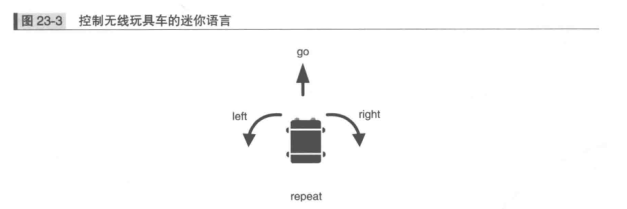
·右转（right）

·左转（1eft）

以上就是可以向玩具车发送的命令。go是前进1米后停止的命令；right是原地向右转的命令；1eft是原地向左转的命令。在实际操作时，是不能完全没有偏差地原地转弯的。为了使问题简单化，我们这里并不会改变玩具车的位置，而是像将其放在旋转桌子上一样，让它转个方向。

如果只是这样，大家可能感觉没什么意思。所以，接下来我们再加一个循环命令。

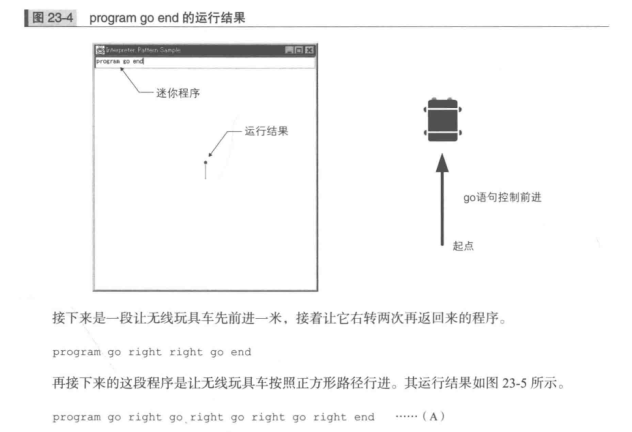
·重复（repeat）以上命令组合起来就是可以控制无线玩具车的迷你语言了。我们会在本章使用迷你语言学习Interpreter模式。

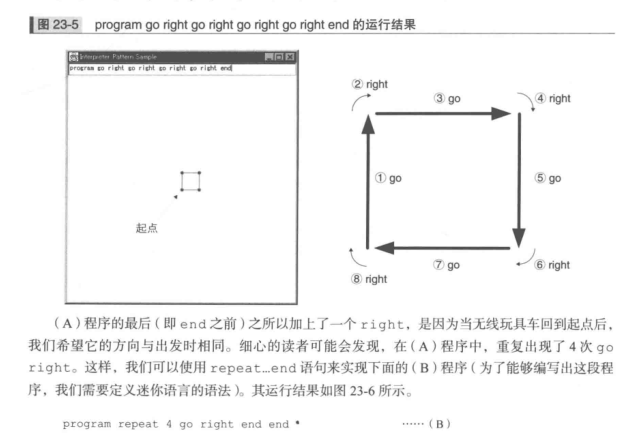


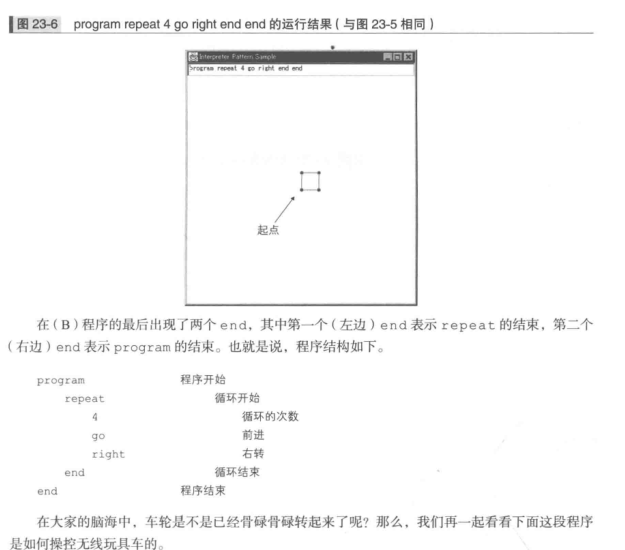
|迷你语言程序示例

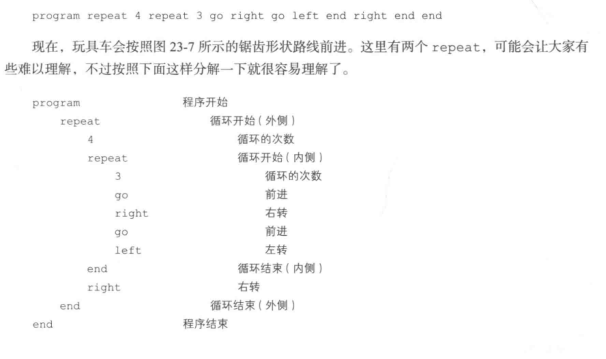
下面我们来看一段用迷你语言编写的迷你程序。下面这条语句可以控制无线玩具车前进（之后停止）。

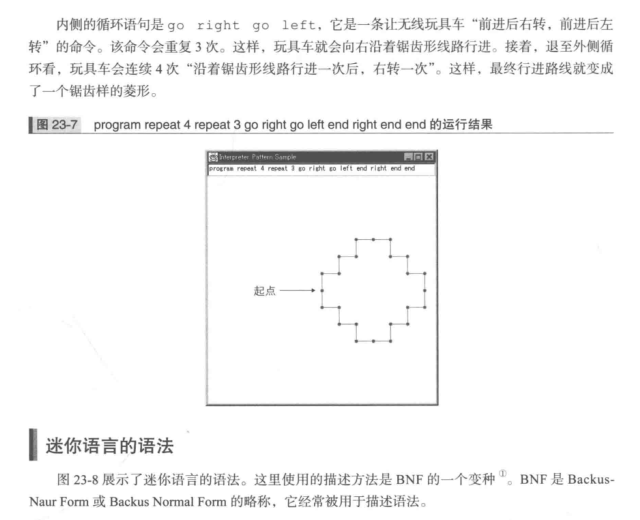
program go end为了便于大家看出语句的开头和结尾，我们在语句前后分别加上了program和end关键字（我们稍后会学习迷你语言的语法）。这个迷你程序的运行结果请参见图23-4（GUI界面是在习题23-1中加入的）。

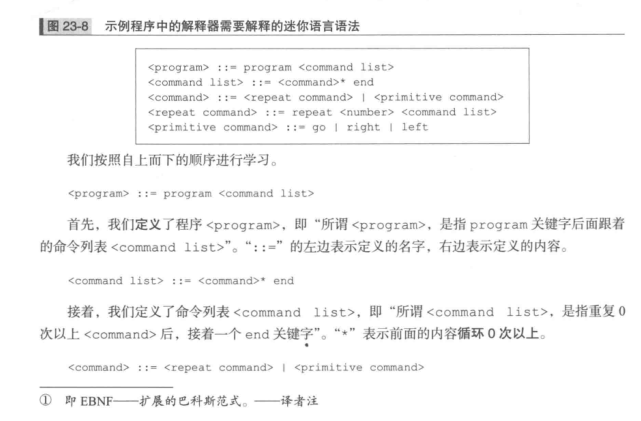


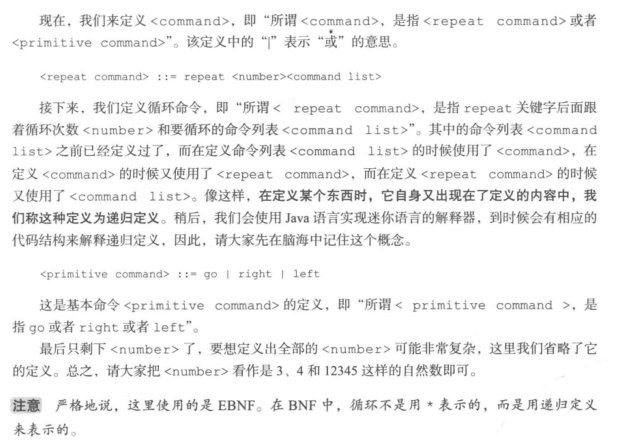












终结符表达式与非终结符表达式

我们先来稍微了解一下语法术语。

前面讲到的像<primitive command>这样的不会被进一步展开的表达式被称为“终结符表达式”（Nonterminal Expression）。我们知道，巴士和列车的终到站被称为终点站，这里的终结符就类似于终点站，它表示语法规则的终点。

与之相对的是，像<program>和<command>这样的需要被进一步展开的表达式被称为“非终结符表达式”。

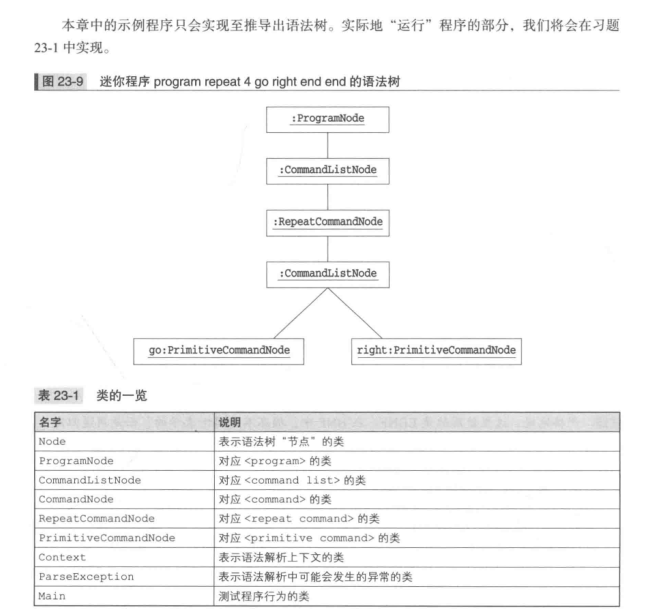
|23.3示例程序

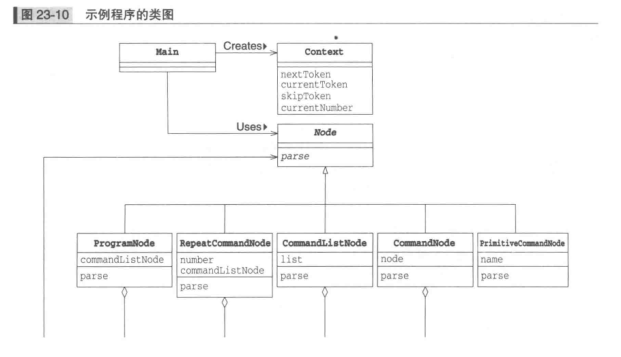
迷你语言的学习至此就结束了，下面我们来看看示例程序。这段示例程序实现了一个迷你程序的语法解析器。

在之前学习迷你程序的相关内容时，我们分别学习了对迷你程序的各个语法部分。像这样将迷你程序当作普通字符分解，然后看看各个部分分别是什么结构的过程，就是语法解析。

例如有如下迷你程序。

program repeat 4 go right end end将这段迷你程序推导成为图23-9中那样的结构（语法树）的处理，就是语法解析。





|23.4Interpreter模式中的登场角色

在Interpreter模式中有以下登场角色。

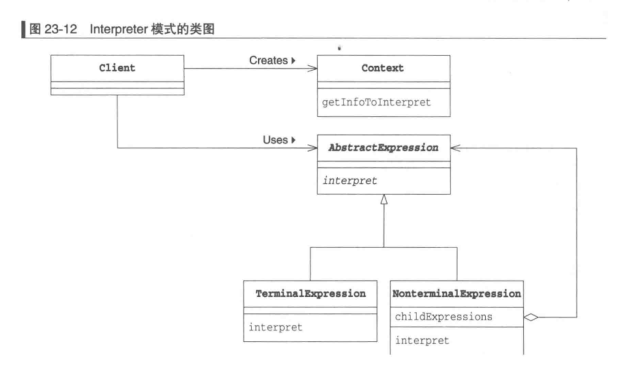
AbstractExpression（抽象表达式）AbstractExpression角色定义了语法树节点的共同接口（APl）。在示例程序中，由Node类扮演此角色。在示例程序中，共同接口（APl）的名字是parse，不过在图23-12中它的名字是interpreter。

◆TerminalExpression（终结符表达式）TerminalExpression角色对应BNF中的终结符表达式。在示例程序中，由PrimitiveCommandNode类扮演此角色。

◆NonterminalExpression（非终结符表达式）NonterminalExpression角色对应BNF中的非终结符表达式。在示例程序中，由ProgramNode类、CommandNode类、RepeatCommandNode类和CommandListNode类扮演此角色。

◆Context（文脉、上下文）Context角色为解释器进行语法解析提供了必要的信息。在示例程序中，由Context类扮演此角色。

◆Client（请求者）为了推导语法树，Client角色会调用TerminalExpression角色和NonterminalExpression角色。在示例程序中，由Main类扮演此角色。



|23.5拓展思路的要点

还有其他哪些迷你语言

在本章中，我们设计了一种操控无线玩具车的迷你语言。当然，这不过是Interpreter模式的一个例子而已，这里我们再列举一些其他的迷你语言。

◆正则表达式

在GoF书（请参见附录E[GoF]）中，作者使用正则表达式（regular expression）作为迷你语言示例。在书中，作者使用Interpreter模式解释了如下表达式，并推导出语法树。

rainings（dogs I cats）·

这个表达式的意思是“在raining后重复出现0次以上dogs或cats”。

◆检索表达式

在Grand书（请参见附录E[Grand]）中，作者讲解了表示单词组合的Litle Language模式。在书中，该模式可以解释如下表达式并推导出语法树。

garlic and not onions这个表达式的意思是“包含garlic但不包含onions”。

·批处理语言

Interpreter模式还可以处理批处理语言，即将基本命令组合在一起，并按顺序执行或是循环执行的语言。本章中的无线玩具车操控就是一种批处理语言。

|跳过标记还是读取标记

在制作解释器时，经常会出现多读了一个标记或是漏读了一个标记的Bug。在编写各个终结符表达式对应的方法时，我们必须时刻注意“进入这个方法时已经读至哪个标记了？出了这个方法时应该读至哪个标记？”

|23.6相关的设计模式

·Composite模式（第ll章）NonterminalExpression 角色多是递归结构，因此常会使用Composie模式来实现NonteminalExpresion角色。

◆Flyweight模式（第20章）有时会使用Flyweight模式来共享TerminalExpression角色。

◆Visitor模式（第13章）在推导出语法树后，有时会使用Visitor模式来访问语法树的各个节点。