行为型模式-解释器模式

概述：
　　在软件构建过程中，如果某一特定领域的问题比较复杂，类似的模式不断重复出现，如果使用普通的编程方式来实现将面临非常频繁的变化；在这种情况下，将特定领域的问题表达为某种语法规则下的句子，然后构建一个解释器来解释这样的句子，从而达到解决问题的目的．给定一个语言，定义它的文法的一种表示，再定义一种解释器，这个解释器用来解释语言中的句子．

示例代码：

In [2]:
```
class playContext:
    message = ''

class expression:
    def interpret(self, context):
        if len(context.message) == 0:
            return
        else:
            msg = context.message.split(' ')
            for j in msg:
                pos = 0
                for k in j:
                    if not k.isdigit():
                        pos += 1
                        continue
                    break
                hexian = j[0:pos]
                jiezou = j[pos:]
                self.execute(hexian, jiezou)
    def execute(self, a, b):
        pass

class guiter_expression(expression):
    def execute(self, a, b):
        s = "和弦是：%s；节奏是：%s" %(a, b)
        print(s)

if __name__ == '__main__':
    context = playContext()
    context.message = 'C53231323 Am53231323 F43231323 G63231323'
    x = guiter_expression()
    x.interpret(context)
```

```
和弦是：C；节奏是：53231323
和弦是：Am；节奏是：53231323
和弦是：F；节奏是：43231323
和弦是：G；节奏是：63231323
```

应用场景：
　　若一个问题重复发生，可以考虑使用解释器模式；这点在数据处理和日志处理过程中使用较多，当数据的需求方需要将数据纳为己用时，必须将数据"翻译"成本系统的数据规格；同样的道理，日志分析平台也需要根据不同的日志格式翻译成统一的"语言"