Scripting com Boo

Criando DSLs internas mais expressivas



Fone: (21) 93174772

MSN: me@juanlopes.net

Login DevMedia: juanplopes@gmail.com

**Juan Lopes**

me@juanlopes.net

Arquiteto de soluções pela Living Consultoria. Iniciou a carreira desenvolvendo aplicações biométricas em C++ e Java. Programa principalmente em C# desde 2007, mas desenvolve também em Ruby e Python. É entusiasta da comunidade Microsoft e participa de projetos open source.

*Sugestão de Ilustração: fantasmas, roteiros, quebra-cabeças*

|  |
| --- |
| **De que se trata o artigo** |
| Este artigo trata sobre a criação de mecanísmos de scripting com Boo, uma linguagem criada pelo brasileiro Rodrigo de Oliveira (também conhecido como Rodrigo Bamboo). É uma linguagem que possui sintaxe e paradigma similares aos do Python, com o benefício de ser estaticamente tipada. |
| **Para que serve** |
| Por ser compilada, Boo tira proveito da maior parte das funcionalidades da Common Language Infrastructure o que a torna tão flexível quanto qualquer outra linguagem do .Net. Aliando-se à extensibilidade do compilador e à sintaxe elegante e sucinta vinda do Python, não é difícil compreender porque ela tem se tornado cada vez mais a linguagem preferida para se desenvolver DSLs de forma rápida e confiável. |
| **Em que situação o tema é útil** |
| Esse tema é especialmente útil quando é necessário alcançar uma expressividade maior ao desenvolver uma linguagem de representação para os sistemas. Boo, apesar de compilada, suporta a maior parte das facilidades de linguagens dinâmicas. Além disso, é uma linguagem que evita repetições desnecessárias, o que faz com que sua representação seja quase sempre direta e legível. |

**Scripting com Boo**

Em muitas situações é necessário desenvolver mecanismos de expressão de certos comportamentos do sistema. Existem formas canônicas de realizar esse tipo de tarefa, como declarativamente por XML ou via banco de dados, mas muitas vezes é preciso alcançar uma expressividade maior. Para esse propósito desenvolvem-se DSLs. Este artigo irá tratar de uma das formas de fazê-lo utilizando a linguagem Boo. Este artigo irá tratar sobre como utilizar Boo para realizar tarefas de scripting para expressividade de problemas de domínio, mostrando alguns exemplos práticos de uso da linguagem.

O .Net é uma plataforma que roda sobre uma máquina virtual. Esta máquina virtual tem o nome de Common Language Runtime permite que as diversas linguagens que compilam em para ela possam ser executadas no mesmo ambiente e se comuniquem através de protocolos comuns. Isso estimula que várias linguagens sejam criadas para essa plataforma.

Boo é uma dessas linguagens. Ela existe desde 2003. Apesar de não ser tão nova, Boo ganhou destaque na comunidade recentemente após Oren Eini (também conhecido como Ayende Rahien), um dos maiores contribuidores do Nhibernate, escrever um livro sobre DSLs em Boo.

A estrutura de um código em Boo lembra muito um código em Python. As duas linguagem tem suporte a diversas funcionalidades parecidas, como duck typing, inferência automática de tipos, generators, funções de primeira classe entre outras coisas que permitem uma maior expressão com a linguagem. Tudo isso sendo uma linguagem estaticamente tipada.

**Nota do Devman**

Duck typing é uma forma de tipagem que verifica a existência de membros num objeto somente no momento em que eles são referenciados em tempo de execução. É uma característica de tipagem dinâmica. Esse nome é oriúndo da famosa frase de James Riley, que é traduzida livremente como “se eu vejo uma ave que anda como um pato, nada como um pato e faz barulho de pato, eu chamo essa ave de pato”. Fazendo alusão de que se um objeto responde às mesmas mensagens de outro, eles podem ser considerados equivalentes em um determinado contexto, mesmo que seus tipos sejam diferentes.

Além disso, Boo tem uma capacidade que a diferencia da maior parte das linguagens. Seu compilador é feito para ser extensível e facilmente embutido em aplicações, o que a torna perfeita para utilização em mecanismos de extensibilidade por scripting.

Scripting

Cada projeto de software precisa permitir níveis de configuração diferentes. Para certas aplicações, um XML de configuração é suficiente. Outras armazenam complexos parâmetros de configuração em banco de dados. Existem aquelas que permitem tal nível de configuração que é possível modificar completamente o comportamento usando plug-ins.

Uma das alternativas de extensibilidade mais usadas é o scripting. A possibilidade do usuário escrever código que realize tarefas na aplicação. Em geral, é uma técnica aplicada a usuários mais avançados, mas com um pouco de cuidado, é possível desenvolver uma linguagem limpa e expressiva para facilitar a vida de quem tenta customizar a aplicação.

Quando se fala de scripting, geralmente existem alguns desafios. O primeiro deles é a escolha da linguagem a ser usada como base. É possível criar uma linguagem específica para esse propósito ou utilizar uma já existente com as devidas modificações. O segundo desafio é no campo da performance. Dependendo de como os scripts forem ser executados, sua performance pode ser ruim em aplicações de alto desempenho, exigindo níveis de otimização que geralmente não são praticáveis. Já o terceiro desafio é o da legibilidade dos scripts. Se o usuário final vai precisar ser capaz de modificar esses scripts, é preciso que ele saiba entender o que um certo script faz.

Tendo esses problemas em mente, a escolha de Boo torna-se mais natural, pois:

* utilizar uma linguagem que compila em CIL permite que ela se comunique com objetos já existentes e utilize estruturas da própria aplicação sem grandes customizações;
* a performance de uma linguagem compilada é quase sempre melhor que a das interpretadas;
* Boo herda o estilo de Python, uma linguagem que preza pela legibilidade e simplicidade;
* a linguagem permite que suas estruturas sejam customizadas através de extensão do compilador.

Estrutura básica de um código em Boo

Como já foi dito, o código em Boo se parece em muito com um código em Python. A linguagem em si tem o objetivo de ser “amigável ao pulso”, numa referência a menor quantidade de caracteres que precisam ser digitados nos programas da linguagem.

Um exemplo é que Boo não precisa de uma classe “Program”. Nem de um “public static int Main”. Tal qual no Python, qualquer instrução que não estiver dentro de uma classe será executada no começo do programa. Então, o famoso “Hello World” acaba sendo bastante curto.

print "Hello World"

Para quem não está acostumado com Python, o que chama mais atenção à primeira vista num script em Boo é a inexistência de delimitador de fim de bloco. Ou seja, nada de “End If” ou “}”. Toda a demarcação de bloco é feita através de indentação. Isto é, para definir que um grupo de instruções está no mesmo bloco, basta dar a mesma indentação a elas.

if 2+2==4:

print "Um bloco"

else:

print "Outro bloco"

Boo é uma linguagem estaticamente tipada que confia muito na inferência automática de tipos. Isto é, ao escrever um código como o abaixo, o compilador percebe que o retorno da função dois\_mais\_dois é um inteiro, mesmo que isso não tenha sido declarado em momento algum. E ao tentar invocar o método Substring, um erro de compilação é lançado, informando que Substring não é um método que exista em “int”.

def dois\_mais\_dois:

return 2+2

quatro = dois\_mais\_dois()

print quatro.Substring(2)

Além disso, muitas outras construções são semelhantes às respectivas em Python, como, por exemplo, os generators. No código abaixo, é gerada uma enumeração com todos os pares entre 0 e 1000. Entretanto, ela somente será avaliada quando for necessário. O mais tarde possível. No caso, ao exibir, o código obtem somente os cinco primeiros itens dessa enumeração. É importante perceber também o uso do método Take, do Linq to Objects.

pares = i for i in range(1000) if i % 2 == 0

print(join(pares.Take(5)))

**Nota do Devman**

Generators são estruturas presentes na linguagem Python (e em outras também) que permite criar enumerações avaliadas tardiamente. É possível criar um generator utilizando uma sintaxe com expressões “for .. in ... if/unless” ou utilizando generator methods, que possui sintaxe e funcionamento bem parecidos com os iterators do C#.

Uma linguagem expressiva

