Programação Funcional

Para leigos

John Paul Mueller

ALTA BOOKS
EDITORA
Rio de Janeiro, 2019

Sumário Resumido

1 Guia Rápido da Programação Funcional

NESTA PARTE...

Descubra o paradigma de programação funcional.

Entenda em que a programação funcional difere.

Obtenha e instale o Python.

Obtenha e instale o Haskell.

NESTE CAPÍTULO

- Explorando a programação funcional
- Programando no modo funcional
- Encontrando uma linguagem adequada às suas necessidades
- Localizando os recursos da programação funcional

Capítulo **1**

Introdução à Programação Funcional

Este livro não aborda uma linguagem de programação específica, mas um paradigma de programação. *Paradigma* é uma estrutura que expressa certo conjunto de suposições, conta com modos particulares de refletir sobre os problemas e usa determinadas metodologias para resolvê-los. Como consequência, este livro de programação é diferente, porque não determina qual linguagem usar. Pelo contrário, foca os problemas que precisam ser resolvidos. A primeira parte deste capítulo analisa como o paradigma de programação funcional cumpre essa tarefa, e a segunda destaca como tal paradigma difere dos outros que você pode ter usado.

A orientação matemática da programação funcional representa a impossibilidade de utilizá-la para criar um aplicativo; você deve resolver problemas matemáticos simples ou inventar cenários *hipotéticos* para testar. Como a programação funcional é única em sua abordagem de solução de problemas, é possível imaginar como realiza seus objetivos. A terceira parte fornece uma rápida visão geral de como usar o paradigma de programação funcional para fazer

vários tipos de tarefas (inclusive o desenvolvimento tradicional); e a quarta mostra como algumas linguagens seguem um caminho puro até esse objetivo e outras, um impuro. Isso não quer dizer que as que seguem o caminho puro são mais perfeitas do que as do caminho impuro; são simplesmente diferentes.

Finalmente, este capítulo também analisa recursos online que você vê mencionados em outras áreas deste livro. O paradigma de programação funcional é popular para resolver certos problemas. Esses recursos ajudam a descobrir as particularidades de como as pessoas usam a programação funcional e por que acham que é um bom método de resolver problemas. O mais importante é que descobrirá que muitas pessoas que contam com esse paradigma não são desenvolvedores. Portanto, se você não for desenvolvedor, poderá achar que já está em boa companhia ao escolher esse paradigma para atender às suas necessidades.

Definindo a Programação Funcional

A programação funcional tem objetivos e abordagens um pouco diferentes dos outros paradigmas. Os objetivos definem o que o paradigma de programação funcional está tentando fazer ao produzir as abordagens usadas pelas linguagens que a suportam. Porém os objetivos não especificam certa implementação; isso é competência das linguagens individuais.


LEMBRE-SE

A principal diferença entre o paradigma de programação funcional e os outros é que os programas funcionais usam funções matemáticas no lugar de instruções para expressar ideias. Essa diferença significa que, em vez de escrever um conjunto preciso de etapas para resolver um problema, você usará funções matemáticas e não se preocupará com o modo de a linguagem realizar a tarefa. Em alguns aspectos, isso torna as linguagens que suportam esse paradigma parecidas com aplicativos como MATLAB. É claro que com o MATLAB há uma interface do usuário, o que reduz o tempo de aprendizado. Mas a conveniência da interface do usuário implica a perda de capacidade e flexibilidade que as linguagens funcionais oferecem. Usar essa abordagem para definir um problema requer um estilo de *programação declarativa*, usada com outros paradigmas e linguagens, como a Linguagem de Consulta Estruturada (SQL) para o gerenciamento de bancos de dados.

Em oposição aos outros paradigmas, o de programação funcional não mantém o estado. O uso do *estado* permite controlar os valores entre as chamadas da função. Outros paradigmas usam o estado para produzir resultados variáveis com base no ambiente, como determinar o número de objetos existentes e fazer algo diferente quando o número de objetos é zero. Assim, chamar uma função do programa funcional sempre produz o mesmo resultado, dado certo conjunto de entradas, tornando os programas funcionais mais previsíveis do que aqueles que suportam o estado.

Como os programas funcionais não mantêm o estado, os dados com os quais eles trabalham também são *imutáveis*, significando que você não pode mudá-los. Para mudar o valor de uma variável, é necessário criar uma nova. Mais uma vez, isso torna os programas funcionais mais previsíveis do que as outras abordagens e também pode torná-los mais fáceis de executar em vários processadores. As seções a seguir dão informações extras sobre como o paradigma de programação funcional difere dos outros.

Entendendo os objetivos

A *programação imperativa*, o tipo de programação que a maioria dos desenvolvedores fez até o momento, é parecida com uma linha de montagem, na qual os dados se movem em uma série de etapas, em uma ordem específica, para produzir determinado resultado. O processo é fixo e rígido, e a pessoa que implementa o processo deve criar uma nova linha de montagem sempre que um aplicativo requerer um novo resultado. A programação orientada a objetos (OOP) apenas modulariza e oculta as etapas, mas o paradigma subjacente é igual. Mesmo com a modularização, a OOP geralmente não permite uma reorganização do código do objeto de modos não antecipados por causa das interdependências inerentes do código.


LEMBRE-SE

A programação funcional se livra das interdependências substituindo os procedimentos por funções puras, que requerem o uso do estado imutável. Como consequência, a linha de montagem não existe mais; um aplicativo pode manipular os dados usando as mesmas metodologias da matemática pura. A aparente restrição do estado imutável fornece meios que permitem a uma pessoa que entenda a matemática de uma situação também criar um aplicativo para fazer os cálculos.

Usar funções puras cria um ambiente flexível, no qual a ordem do código depende da matemática subjacente. Essa matemática modela um ambiente real, e, quando nosso entendimento do ambiente muda e evolui, o modelo matemático e o código funcional podem mudar com ele, sem os problemas normais e delicados que fazem o código imperativo falhar. Modificar o código funcional é mais rápido e menos propenso a erros porque a pessoa que implementa a mudança deve entender apenas a matemática e não precisa saber como o código subjacente funciona. E mais, aprender a criar um código funcional pode ser mais rápido, contanto que se compreenda o modelo matemático e sua relação com o mundo real.

A programação funcional também adota várias abordagens únicas de codificação, como a capacidade de passar uma função para outra como entrada. Essa capacidade permite mudar o comportamento do aplicativo de um modo previsível, que não é possível usando outros paradigmas de programação. No decorrer deste livro, você encontrará outros benefícios de usar a programação funcional.

Usando a abordagem pura

As linguagens de programação que usam a abordagem pura para o paradigma de programação funcional contam muito com os princípios do cálculo lambda. E mais, uma linguagem com abordagem pura permite usar apenas as técnicas da programação funcional para que o resultado seja sempre um programa funcional. A linguagem com abordagem pura usada neste livro é o Haskell porque fornece a implementação mais pura, segundo os artigos encontrados no Quora, em https://www.quora.com/What-are-the-most-popular-and-powerful-functional-programming-languages [conteúdo em inglês]. O Haskell também é uma linguagem relativamente popular, segundo o índice TIOBE (https://www.tiobe.com/tiobe-index/ — conteúdo em inglês). Outras linguagens com abordagem pura incluem Lisp, Racket, Erlang e OCaml.


CUIDADO

Como em muitos elementos da programação, há fortes opiniões em relação à possibilidade de certa linguagem de programação se qualificar a um status puro. Por exemplo, muitas pessoas considerariam o JavaScript uma linguagem pura, mesmo que seja não tipado. Outras acham que as linguagens declarativas específicas do domínio, como o SQL e o Lex/Yacc, qualificam-se ao status puro mesmo que não sejam linguagens de programação gerais. Simplesmente ter elementos da programação funcional não qualifica uma linguagem como adepta da abordagem pura.

Usando a abordagem impura

Muitos desenvolvedores chegaram a ver os benefícios da programação funcional. Mas eles não querem abrir mão das vantagens da linguagem que já utilizam, portanto usam uma linguagem que mistura os recursos funcionais com um dos outros paradigmas da programação (como descrito na seção "Considerando Outros Paradigmas da Programação", adiante). Por exemplo, você pode encontrar recursos da programação funcional em linguagens como C++, C# e Java. Ao trabalhar com uma linguagem impura, é preciso ter cuidado, porque o código não funcionará de maneira unicamente funcional e os recursos que você pode achar que funcionarão de um jeito, na verdade, funcionarão de outro. Por exemplo, em algumas linguagens não é possível passar uma função para outra.


DICA

Pelo menos uma linguagem, Python, é projetada desde o início para suportar vários paradigmas da programação (veja https://blog.newrelic.com/2015/04/01/python-programming-styles/ para obter detalhes). Na verdade, alguns cursos online fazem questão de ensinar esse aspecto em particular do Python como um benefício especial (veja https://www.coursehero.com/file/p1hkiub/Python-supports-multiple-programming-paradigms-including-object-oriented/). O uso de vários paradigmas da programação torna o Python bem flexível, mas também leva a reclamações e defesas (veja http://archive.oreilly.com/pub/post/pythons_weak_functional_progra.html

como um exemplo) [conteúdos em inglês]. Os motivos para este livro contar com o Python para demonstrar a abordagem impura para a programação funcional é que ele é popular e flexível, além de ser fácil de aprender.

Considerando Outros Paradigmas de Programação

Você pode achar que existem apenas alguns paradigmas de programação além do explorado neste livro, mas o mundo do desenvolvimento está repleto deles. Isso porque duas pessoas não pensam 100% da mesma forma. Cada paradigma representa uma abordagem diferente, que compõe o quebra-cabeças de passar uma solução para os problemas usando determinada metodologia e ainda fazer suposições sobre coisas como a capacidade do desenvolvedor e o ambiente de execução. Na verdade, é possível encontrar sites inteiros que discutem o problema, como em `http://cs.lmu.edu/~ray/notes/paradigms/` [conteúdo em inglês]. Por incrível que pareça, algumas linguagens (como o Python) misturam e combinam paradigmas compatíveis para criar uma maneira inteiramente nova de realizar tarefas com base no que aconteceu no passado.


LEMBRE-SE

As seções a seguir descrevem apenas quatro desses outros paradigmas. Eles não são melhores nem piores do que qualquer outro, mas representam correntes comuns de pensamento. Muitas linguagens no mundo usam hoje apenas esses quatro paradigmas, portanto suas chances de encontrá-los são bem altas.

Imperativo

A programação imperativa adota uma abordagem passo a passo para realizar uma tarefa. O desenvolvedor fornece comandos que descrevem precisamente como realizar a tarefa do início ao fim. Durante o processo de execução dos comandos, o código também modifica o estado do aplicativo, incluindo seus respectivos dados. O código é executado por completo. Um aplicativo imperativo imita bem o hardware do computador, que executa o código de máquina. *Código de máquina* é o menor conjunto de instruções que você pode criar e é imitado nas primeiras linguagens, como o assembler.

Procedural

A programação procedural implementa a programação imperativa, mas adiciona uma funcionalidade, como blocos de código e procedimentos para dividir o código. O compilador ou o interpretador ainda acaba produzindo o código de máquina que é executado passo a passo, mas o uso de procedimentos facilita que o desenvolvedor siga o código e entenda como funciona. Muitas linguagens

procedurais fornecem um modo de desmontagem, no qual você vê a correspondência entre a linguagem de nível mais alto e o assembler subjacente. Exemplos de linguagens que implementam o paradigma procedural são C e Pascal.

PAPO DE ESPECIALISTA

As primeiras linguagens, como o Basic, usavam o modelo imperativo porque os desenvolvedores que as criaram estavam acostumados a trabalhar com o hardware do computador. Mas os usuários Basic geralmente enfrentavam um problema chamado *código espaguete*, que fazia os aplicativos grandes parecerem um único bloco. A menos que você fosse o desenvolvedor do aplicativo, seguir a lógica dele geralmente era difícil. Como consequência, as linguagens que seguem o paradigma procedural estão um passo à frente das linguagens que seguem apenas o paradigma imperativo.

Orientado a objetos

O paradigma procedural realmente facilita a leitura do código. Mas a relação entre o código e o hardware subjacente ainda dificulta relacionar o que o código faz com o mundo real. O paradigma orientado a objetos usa o conceito de objetos para ocultar o código, porém o mais importante é facilitar a modelagem do mundo real. Um desenvolvedor cria objetos de código que imitam os objetos reais que eles copiam. Esses objetos incluem propriedades, métodos e eventos para permitir ao objeto se comportar de certa maneira. Exemplos de linguagens que implementam o paradigma orientado a objetos são C++ e Java.

LEMBRE-SE

As linguagens que implementam o paradigma de orientação a objetos também implementam os paradigmas procedural e imperativo. O fato de que os objetos ocultam o uso desses outros paradigmas não significa que um desenvolvedor não escreveu o código para criar o objeto usando os paradigmas mais antigos. Assim, o paradigma orientado a objetos ainda conta com o código que modifica o estado do aplicativo, mas também pode permitir modificar os dados variáveis.

Declarativo

A programação funcional de fato implementa o paradigma de programação declarativa, mas os dois paradigmas são separados. Outros paradigmas, como a programação lógica, implementada pela linguagem Prolog, também suportam o paradigma de programação declarativa. O resumo do que a programação declarativa faz é o seguinte:

- Descreve o que o código deve fazer, em vez de como fazê-lo.
- Define as funções que são transparentes de modo referencial (sem efeitos colaterais).
- Fornece uma correspondência clara para a lógica matemática.

Usando a Programação Funcional para Realizar Tarefas

É essencial lembrar que a programação funcional é um paradigma, o que significa que não tem uma implementação. A base da programação funcional é o cálculo lambda (`https://brilliant.org/wiki/lambda-calculus/` — conteúdo em inglês), que é uma abstração matemática. Logo, quando quiser realizar tarefas usando o paradigma de programação funcional, você buscará uma linguagem de programação que implementa a programação funcional de uma maneira que atenda às suas necessidades. (A próxima seção, "Descobrindo as Linguagens que Suportam a Programação Funcional", descreve em detalhes as linguagens disponíveis.) De fato, é possível até realizar tarefas da programação funcional em sua linguagem atual sem perceber. Sempre que você cria e usa uma função lambda, provavelmente usa técnicas da programação funcional (de um modo impuro, pelo menos).

Além de usar funções lambda, as linguagens que implementam o paradigma de programação funcional têm outros recursos em comum. Eis uma visão geral rápida desses recursos:

- **Funções de primeira classe e funções de alta ordem:** As funções de primeira classe e as de alta ordem permitem fornecer uma função como entrada, como seria ao usar uma função de alta ordem no cálculo.
- **Funções puras:** Uma função pura não tem efeitos colaterais. Ao trabalhar com uma função pura, é possível:
  - Remover a função se nenhuma outra contar com sua saída.
  - Obter os mesmos resultados sempre que você chamar a função com certo conjunto de entradas.
  - Inverter a ordem das chamadas para diferentes funções sem nenhuma alteração na funcionalidade do aplicativo.
  - Processar as chamadas da função em paralelo sem nenhuma consequência.
  - Avaliar as chamadas da função em qualquer ordem, supondo que a linguagem inteira não permita efeitos colaterais.
- **Recursão:** As implementações da linguagem funcional contam com a recursão para implementar um loop. Em geral, a recursão trabalha de modo diferente nas linguagens funcionais porque não ocorre nenhuma mudança no estado do aplicativo.
- **Transparência referencial:** O valor de uma variável (um nome pouco adequado, porque não se pode mudar o valor) nunca muda em uma implementação da linguagem funcional porque essa linguagem não tem um operador de atribuição.

LEMBRE-SE

É comum encontrar várias outras considerações para realizar tarefas nas implementações da linguagem de programação funcional, mas essas questões não são consistentes entre as linguagens. Por exemplo, algumas usam uma avaliação estrita (ansiosa), ao passo que outras usam uma não estrita (preguiçosa). Na avaliação estrita, a linguagem verifica por completo a função antes da avaliação. Mesmo quando um termo na função não é usado, um termo com problemas fará a função inteira falhar. Contudo, na avaliação não estrita, a função falhará apenas se o termo com problemas for usado para criar uma saída. As linguagens Miranda, Clean e Haskell implementam a avaliação não estrita.

Várias implementações da linguagem funcional também usam sistemas diferentes, portanto o modo como o computador subjacente detecta um tipo de valor muda entre as linguagens. E mais, cada linguagem suporta o próprio conjunto de estruturas de dados. Essas questões não são bem definidas como parte do paradigma de programação funcional, embora sejam importantes para criar um aplicativo; assim, você deve contar com a linguagem usada para defini-las. Supor uma implementação em particular em qualquer linguagem dada é uma má ideia porque ela não é bem definida como parte do paradigma.

Descobrindo as Linguagens que Suportam a Programação Funcional

Para realmente usar o paradigma de programação funcional, você precisa de uma linguagem que o implemente. Como nos outros paradigmas vistos neste capítulo, as linguagens geralmente falham em implementar toda ideia que o paradigma forneça ou implementam as ideias de modos incomuns. Assim, saber as regras do paradigma e ver como a linguagem selecionada o implementa ajuda a entender melhor os prós e os contras de determinada linguagem. E mais, compreender o paradigma facilita comparar uma linguagem com outra. O paradigma de programação funcional suporta dois tipos de implementação de linguagem, pura e impura, como descrito nas seções a seguir.

Considerando as linguagens puras

Uma linguagem de programação funcional pura é aquela que implementa apenas o paradigma de programação funcional. Isso pode parecer um pouco limitado, mas quando você ler os requisitos na seção "Usando a Programação Funcional para Realizar Tarefas", anteriormente neste capítulo, descobrirá que a programação funcional e os paradigmas da programação sem nenhuma relação com o paradigma imperativo (que se aplica à maioria das linguagens disponíveis hoje) são mutuamente excludentes.

Tentar descobrir qual linguagem implementa melhor o paradigma de programação funcional é praticamente impossível porque cada pessoa tem uma opinião sobre o assunto. Você encontra uma lista das 21 implementações da linguagem de programação funcional, com seus prós e contras, em `https://www.slant.co/topics/485/~best-languages-for-learning-functional-programming` [conteúdo em inglês].

Considerando as linguagens impuras

Provavelmente, o Python é o exemplo típico de linguagem impura, porque suporta muitos estilos de codificação. Dito isso, a flexibilidade que o Python fornece é um motivo para que as pessoas gostem tanto de usá-lo: é possível codificar em qualquer estilo que você precise no momento. A definição de uma linguagem impura é que ela não segue totalmente as regras do paradigma de programação funcional (ou, pelo menos, não o bastante para chamá-la de pura). Por exemplo, permitir qualquer modificação no estado do aplicativo desqualificaria imediatamente a linguagem em consideração.


LEMBRE-SE

Um dos motivos mais comuns e menos compreendidos para desqualificar uma linguagem como sendo uma implementação pura do paradigma de programação funcional é a falta do suporte de funções puras. Uma função pura define uma relação específica entre entradas e saídas sem efeitos colaterais. Toda chamada para uma função pura com entradas específicas sempre armazena precisamente a mesma saída, tornando as funções puras extremamente confiáveis. Mas alguns aplicativos realmente contam com os efeitos colaterais para funcionar corretamente, tornando a abordagem pura bem estrita em alguns casos. Os Capítulos 4 e 5 fornecem os detalhes sobre a questão das funções puras. Você também encontra mais no artigo em `http://www.onlamp.com/2007/07/12/introduction-to-haskell-pure-functions.html` [conteúdo em inglês].

Encontrando a Programação Funcional Online

A programação funcional ficou extremamente popular porque ela resolve muitos problemas. Como tratado neste capítulo, ela também tem alguns limites, como a incapacidade de usar dados variáveis; porém, para a maioria das pessoas, os prós superam os contras nas situações que permitem definir um problema usando a matemática pura. (A falta de suporte dos dados variáveis também tem prós, como você descobrirá mais adiante, como a capacidade de fazer o multiprocessamento com mais facilidade.) Dito tudo isso, é ótimo ter recursos ao descobrir um paradigma de programação. Este livro é seu primeiro recurso, mas um único livro não pode analisar tudo.

DICA

Sites online, como Kevin Sookochef [os conteúdos a seguir estão em inglês] (https://sookocheff.com/post/fp/a-functional-learning-plan/) e Wildly Inaccurate (https://wildlyinaccurate.com/functional-programming-resources/), oferecem ótimos recursos úteis. O Hacker News (https://news.ycombinator.com/item?id=16670572) e o Quora (https://www.quora.com/What-are-good-resources-for-teaching-children-functional-programming) também podem ser excelentes recursos. O site Quora, referenciado, é especialmente importante porque fornece informações que são úteis para iniciar as crianças na programação funcional. Um aspecto essencial de usar sites online é assegurar que eles sejam atuais. O recurso não deve ter mais de dois anos; do contrário, você estará recebendo material desatualizado.

Às vezes, é possível encontrar vídeos úteis online. Naturalmente, você pode encontrar inúmeros vídeos com qualidade variada no YouTube (https://www.youtube.com/results?search_query=Functional+Programming), mas não sites desacreditados, como o tinymce (https://go.tinymce.com/blog/talks-love-functional-programming/). Como a programação funcional é um paradigma, e a maioria desses vídeos foca uma linguagem específica, é preciso escolher os vídeos assistidos com cuidado ou terá uma visão distorcida do que o paradigma pode fornecer (em comparação com a linguagem).

CUIDADO

Um recurso tendencioso são os tutoriais. Por exemplo, o tutorial em https://www.hackerearth.com/practice/python/functional-programming/functional-programming-1/tutorial/ é só sobre Python, que, como mencionado nas seções anteriores deste capítulo, é uma implementação impura. Do mesmo modo, até os criadores sérios de tutoriais, como o Tutorials Point (https://www.tutorialspoint.com/functional_programming/functional_programming_introduction.htm), têm problemas com esse assunto porque não é possível demonstrar um princípio sem uma linguagem. Um tutorial não pode ensinar sobre um paradigma, pelo menos não facilmente e não muito além de uma abstração. Assim, ao ver um tutorial, mesmo que tenha como objetivo fornecer uma visão imparcial da programação funcional (como em https://codeburst.io/a-beginner-friendly-intro-to-functional-programming-4f69aa109569), conte com um certo nível de distorção, porque provavelmente os exemplos aparecerão usando um subconjunto das linguagens disponíveis.