Парадигми за програмиране езици за функционално програмиране

Парадигма на програмиране

Парадигмата на програмиране, парадигма за програмиране или програмна парадигма представлява фундаменталния стил на програмиране. Има множество програмни парадигми, но основните сред тях са: обектно ориентирано, императивно, функционално и декларативно

Парадигмата на програмиране предоставя (и определя) начина, по който програмистът гледа на изпълнението на програмата. Например, в обектно ориентираното програмиране програмистът може да разглежда програмата като съвкупност от взаимодействащи си обекти, докато при функционалното програмиране програмата може да бъде разглеждана като последователност от изчисления на функции, които нямат свои състояния.

Различните програмни езици препоръчват различни парадигми на програмиране. Някои езици са проектирани да поддържат някоя определена парадигма на програмиране – SmallTalk и Java поддържат обектно ориентираното програмиране, докато Lisp поддържа функционалното програмиране. Някои езици за програмиране поддържат повече от една парадигма – C++ поддържа и обектно ориентирано програмиране, и процедурно програмиране.

Много парадигми на програмиране са познати повече с това какво забраняват, от колкото с това какво препоръчват. Например функционалното програмиране забранява използването на странични ефекти, структурното програмиране забранява употребата на goto. Заради това новите парадигми обикновено се приемат като твърде сурови и ограничаващи от програмистите, свикнали вече с някакъв предишен стил на програмиране. Но трябва да се отбележи, че избягването на определени техники може да подпомогне доказването на коректността на програмата или просто да улесни разбирането на поведението ѝ, като при това не ограничава приложимостта на програмния език.

Отношението между парадигмите на програмиране и програмните езици може да бъде доста сложно, тъй като един език може да поддържа повече от една парадигма. Например C++ е проектиран да поддържа елементи от процедурното програмиране, обектно ориентираното програмиране, обектно базираното програмиране и родовото програмиране. Един C++ програмист може да напише чисто процедурна програма, или чисто обектно ориентирана програма, а може да напише и програма, която обединява елементи и от двете парадигми.

Функционално програмиране

В компютърните науки функционално програмиране е парадигма за програмиране – стил за изграждането на структурата и елементите на компютърни програми, който третира като изчислява оценката на математически функции и избягва променящите състоянието непостоянни данни. Това е декларативна парадигма за програмиране, което означава, че програмирането се извършва с изрази. Във функционален код, изходната стойност на функция зависи само от аргументите, които са вложени по време на функцията, така призованата функция е два пъти с една и съща стойност за един аргумент Х и тя ще произведе същия резултат (Х) всеки път. Премахване на нежелани реакции, т.е. промени в състоянието, което не зависи от входовете на функцията, може да направи много по-лесно разбирането и да се предскаже поведението на една програма, което е един от основните мотиви за развитието на функционално програмиране.

Функционално програмиране има своите корени в ламбда пресмятането, което е официална система, разработена през 1930 г., за да разследва изчислимост на дефиниция на функция, приложение функция и рекурсия. Много функционални езици за програмиране могат да бъдат разглеждани като изградени върху ламбда пресмятането. Друга добре позната декларативна парадигма за програмиране е логическото програмиране което, се основава на релацията.

За разлика, императивното програмиране променя състоянието, чрез команди в началния език, най-прост пример за това е задачата. Императивното програмиране наистина има функции, не в математическия смисъл, а в смисъл на подпрограми. Те могат да имат странични ефекти, които могат да променят стойността на програмата. Функции без стойности за връщане могат да доведат до това. Поради това, че те нямат референтна прозрачност, т.е. същия израз в друг език може да доведе до различни стойности по различно време в зависимост от състоянието на изпълняващата програмата.

Функционални програмни езици, особено изцяло функционални такива като Hope и Rex, са използвани повече в академичните среди, отколкото в развитието на търговски софтуер. Въпреки това, видни функционални езици за програмиране като Common Lisp, Scheme, Clojure, Language Wolfram Language, Racket, Erlang, OCaml, Haskell, и F# са били използвани в промишлени и търговски заявления от голямо разнообразие от организации. Функционално програмиране се подкрепя и в някои специфични програмни езици за домейн като R, J, K и (programming language from Kx Systems) Q от Kx Systems (финансов анализ), XQuery / XSLT (XML), и Opal. Широко разпространени специфични програмни езици за домейн като SQL и Lex / Yacc използват някои елементи на функционалното програмиране.

Програмиране на функционален стил може да се осъществи в езици, които не са специално предназначени за функционално програмиране. Например, императивния Perl език за програмиране е бил обект на една книга, описваща как да се прилагат концепции на функционалното програмиране. Това е вярно и за езика PHP. C # 3.0 и Java 8 добавят конструкции за улесняване на функционалния стил. Езикът Julia също предлага функционални възможности за програмиране. Интересен случай е този на Scala – често се пише във функционален стил, но присъствието на странични ефекти и непостоянни състояния го провежда в сивата зона между императивни и функционални езици.

Функционално програмиране в нефункционални езици

Възможно е да се използва функционален стил на програмиране на езици, които не са традиционно смятани за функционални езици. Например, и D (език за програмиране) и Fortran 95 изрично подкрепят чисти функции.

JavaScript и Python имат първокласни функции още от самото начало. Armit Prem добави поддръжка на Python за „ламбда“, „карта“, „намаляване“ и „филтър“ през 1994 г., както и закриване (closure) в Python 2.2, макар Python 3 изхвърли „намаляване“-то, в functools стандартна модул библиотека. Първокласни функции са въведени в други масови езици като PHP 5.3, Visual Basic 9, C# 3.0 и C++11.

В Java, анонимните класове понякога могат да бъдат използвани, за да се симулира затваряне. Обаче, анонимни класове не винаги са подходящи заместители на затваряне, защото те имат по-ограничени възможности. Java 8 поддържа ламбда изрази като заместител на някои анонимни класове. Въпреки това, наличието на проверени изключения в Java може да се направи функционалното програмиране неудобно, защото може да се наложи да се хванат проверени изключения и след това ги rethrow-ваме, проблем, който не се среща в други JVM езици, които нямат проверени изключения (Exceptions), като като Scala.

В C#, анонимни класове не са необходими, тъй като затваряне и ламбда се поддържат напълно. Библиотеки и езикови разширения за non muteable структури от данни се разработват, за да се подпомогне функционалното програмиране в C#.

Много обектно-ориентирани модели за проектиране са изразими във функционално програмиране: например, модел на стратегия (strategy pattern) просто диктува използването на функция за по-висш порядък и модел на посетител (visitor pattern) приблизително съответства на катаморфизма или сгъвката.

По същия начин, идеята за неизменни (immutable) данни от функционално програмиране често е включена в императивни езици за програмиране, например кортеж в Python, която е неизменен масив.