Знакомство с функциональным программированием

Лектор: Артамонов Юрий Николаевич

Университет "Дубна"
филиал Котельники

Основной тезис - «объяснить, что сделать»

Для решения задачи на компьютере в большинстве случаев приходится на одном из языков программирования объяснять «железному другу» как это сделать. Однако человека не покидает мечта другого стиля взаимодействия с компьютером — лишь описать что нужно сделать, так, чтобы цепочки кода «как сделать» порождались автоматически. В общем случае эта мечта, по крайней мере сейчас, неосуществима, но есть важные области, где удалось достичь прогресса. Стиль программирования «как сделать» получил название процедурное или императивное программирование, а «что сделать» - декларативное программирование. Функциональное и логическое программирование относят к декларативному стилю.

Общие сведения о функциональном программировании

Функциональное программирование явилось практической реализацией математических теорий λ-исчисления (Алонзо Чёрч, 1936) и комбинаторной логики (Моисей Исаевич Шейнфинкель и Хаскелл Карри). λ-исчисление возникло как один из подходов к формализации понятия алгоритм через безымянные функции и рекурсивные вызовы. Поэтому языки функционального программирования обращаются с вычислительными процессами как с обычными математическими функциями. На первый взгляд, может показаться, что в этом нет ничего нового и непонятно, какую это может дать выгоду.

Свойства программ функционального стиля

Тем не менее использование функционального подхода всегда наделяет программу следующими свойствами:

- лаконичность кода;
- возможность более гибкой модификации программы (ведь вся программа это совокупность независимых описаний функций, расположенных в произвольном порядке, которые могут порождать абстракции друг от друга);
- способность воспринимать данные как программу (таким образом программа может порождать программу для себя самой);
- принципиальная возможность распараллеливания вычислений;
- возможность «ленивых» вычислений;
- отсутствие побочных эффектов.

Пояснение некоторых свойств

В наличии этих свойств можно убедиться лишь начав программировать на практике. Формулировки каждого из этих свойств должны быть понятны, и только два последних требуют пояснений. Свойство «отсутствие побочных эффектов» является следствием обращения с вычислительными процессами как с обычными математическими функциями. Допустим, мы определили некоторую функцию F(x) в процедурном языке программирования. Возможна ли такая ситуация, когда эта функция принимает разные значения от одного и того же аргумента. С точки зрения математики было бы абсурдно получить два выражения — F(3)=9, F(3)=2. Однако в программировании такое возможно, например, F(x)=X+a, где a — глобальная переменная, значение которой может меняться в основном теле программы. Может показаться, что мы всегда сможем, если захотим, программировать без побочных эффектов. Но процедурные языки пронизаны побочными эффектами, начиная с простого присваивания s:=s+1, заканчивая обработкой исключений.

Понятие побочного эффекта

Таким образом, побочный эффект функции — возможность в процессе выполнения своих вычислений чтения и модификации значений глобальных переменных, осуществления операций ввода/вывода, реагирования на исключительные ситуации, вызова их обработчиков. Поэтому, если вызвать такую функцию дважды с одним и тем же набором значений входных аргументов, может случиться так, что в качестве результата вычисляются разные значения. Такие функции называются недетерминированными функциями с побочными эффектами. В чистом функциональном программировании объекты можно только создавать, там нет переменных в традиционном понимании и им нельзя переприсвоить новое значение, их уничтожением автоматически занимается так называемый уборщик мусора. Все это следствие того, что в чистых функциональных программах нет понятия изменения и нет понятия времени.

Чистое функциональное программирование

Термин чистое функциональное программирование также употребляется специально. Дело в том, что большинство реальных функциональных языков все-таки используют побочные эффекты, но предлагают механизмы эффективного разделения чистого функционального подхода с сопутствующими свойствами от императивного стиля.

Ленивые вычисления

Для объяснения, что такое «ленивые вычисления» представим ситуацию, когда в программе производятся манипуляции с бесконечной суммой чисел. В обычном случае, прежде чем подставлять бесконечную сумму в какое-либо выражение программа попытается ее упростить — вычислить, но с бесконечной суммой это сделать невозможно, а значит подставить ее куда либо не удается. Но мы может отложить вычисление до требуемого момента, это и есть ленивые вычисления. В лямбда исчислении ленивые вычисления инициируются так называемым нормальным порядком редукций, вычисления с максимальным упрощением перед подстановкой называются аппликативным порядком редукций. Большинство функциональных языков реализуют аппликативный порядок редукций, однако в них легко можно добиться и нормального порядка.

Параллелизм

Ещё одним важным преимуществом чистых функциональных языков является параллелизм. Раз все функции для вычислений используют только свои параметры, можно организовать вычисление независимых функций в произвольном порядке или параллельно, на результат вычислений это не повлияет. Параллелизм может быть организован не только на уровне компилятора языка, но и на уровне архитектуры технических средств. Существуют экспериментальные компьютеры, основанные на подобных архитектурах, например Lisp-машина. Лисп-машина — универсальная вычислительная машина, архитектура которой оптимизирована для эффективного выполнения программ на языке Лисп. Несмотря, что Лисп-машины никогда не были широко распространены, многие распространённые ныне идеи и программные технологии были впервые разработаны с помощью Лисп-машин.

Некоторые функциональные языки

В настоящее время наиболее распространены следующие языки функционального программирования:

- Haskell
- F#
- Scala
- Erlang
- LISP

Haskell

Haskell (русск. Хаскелл или Хаскель) — функциональный язык программирования. Является одним из самых распространённых нестрогих языков программирования. Имеет развитую систему типизации. Последний стандарт языка, ставший стандартом функционального программирования — Haskell-98. Берёт своё начало из языка Miranda, который был разработан Дэвидом Тёрнером в качестве стандартного функционального языка. Назван по имени математика Хаскелла Карри.

F#

Это мультипарадигмальный язык программирования из семейства языков .NET Framework, поддерживающий функциональное программирование в дополнение к императивному (процедурному) и объектно-ориентированному программированию. Структура F# во многом схожа со структурой OCaml с той лишь разницей, что F# реализован поверх библиотек и среды исполнения .NET. Язык был разработан Доном Саймом (англ. Don Syme) в Microsoft Research в Кембридже.

Scala

Scala — мультипарадигмальный язык программирования, спроектированный кратким и типобезопасным для простого и быстрого создания компонентного программного обеспечения, сочетающий возможности функционального и объектно-ориентированного программирования. Scala-программы во многом похожи на Java-программы, и могут свободно взаимодействовать с Java-кодом. Язык включает единообразную объектную модель — в том смысле, что любое значение является объектом, а любая операция — вызовом метода. При этом является также функциональным языком в том смысле, что функции — это полноправные значения.

Erlang

Erlang — функциональный язык программирования с сильной динамической типизацией, предназначенный для создания распределённых вычислительных систем. Разработан и поддерживается компанией Ericsson. Язык включает в себя средства порождения параллельных легковесных процессов и их взаимодействия через обмен асинхронными сообщениями.

Общие сведения

Лисп (LISP, от англ. LISt Processing — «обработка списков») — семейство языков программирования, основанных на представлении программы системой линейных списков символов, которые являются основной структурой данных языка. Лисп был создан Джоном Маккарти в конце 1950 года с целью решения задач символьного дифференцирования и интегрирования алгебраических выражений. Лисп считается вторым после Фортрана старейшим высокоуровневым языком программирования. Но несмотря, на это он продолжает развиваться.

Общие сведения (продолжение)

Традиционный Лисп имеет строгую динамическую систему типов, содержит императивные свойства, но в общем поощряет функциональную парадигму программирования. Полноценная символьная обработка подразумевает возможность создания любого профиля объектно-ориентированного программирования (то есть реализация полного ООП имеющего тот или иной характер средствами языка). ООП в стандарте Лиспа является платформа CLOS. Неформальная эволюция Лиспа продолжалась долгие годы. Язык Лисп наряду с языком Ada прошёл процесс фундаментальной стандартизации для использования в военном деле и промышленности, в результате чего появился стандарт ANSI Common Lisp. Его реализации существуют для большинства платформ.

Реализации Lisp

Вот несколько реализаций Lisp
свободного распространения:
CLISP - под лицензией GNU GPL, доступен для Windows, Macintosh, и Linux / Unix
OpenMCL - под лицензией GNU LGPL, доступен для Mac, Linux, FreeBSD, Solaris, Windows (in alpha)
CMU Common Lisp - Public License, доступен для Unix и Linux
Steel Bank Common Lisp - производное от CMU Common Lisp (включает компилятор)
коммерческие продукты:
LispWorks - высокого качества и по разумным ценам система для Windows и Linux.
Франц Allegro Lisp - высокое качество и высокая стоимость.
В Лиспе также существует ряд диалектов: Scheme, Clojure, Common Lisp, Emacs Lisp, AutoLisp.

Использование Lisp

Лисп используется как язык сценариев в САПР AutoCAD и текстовом редакторе Emacs. По сути, большая часть Emacs написана на Лиспе, что даёт неограниченные возможности расширения функциональности. В оконном менеджере Sawfish применяется диалект Лиспа Rep, который в значительной степени повторяет диалект Лиспа от Emacs. Sawfish, как и Emacs распространяется на правах GNU GPL. Сегодня Лисп используется во множестве различных проектов: от декодирования генома человека до системы проектирования авиалайнеров. Кроме традиционных научных (Maxima, Axiom, Mathematica) и промышленных применений Лисп используется для высокодинамичных трёхмерных игр (диалект GOAL).