УДК 004.051

Роздобудько С.О.,
Моденов Ю.Б., к.т.н.

ФУНКЦІОНАЛЬНЕ ТА ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ КОНФЛІКТ ВИБОРУ

Національний авіаційний університет

1diller3@mail.ru

За період останніх років, програмування набуло неабиякої популярності серед людей різних, як по характеру, так і по роду їхніх занять. В теперішній час "час технічного прориву" саме на просторах інформаційної індустрії зароджуються новинки, які і дають поштовхи подальшому розвитку. І в саме цей складний період програмісти зайняли одну з ключових позицій в напрямку прогресу

Ключові слова: Програмування, принципи програмування, функції, процедури.

Вступ

Програмування з кожним роком все більше і більше починає нас дивувати. Абсолютно кожен може зробити свій внесок в розвиток цієї індустрії, адже колись малопопулярний напрямок, так би мовити напрямок "мистецтва", став дуже популярним в теперішній час напрямком зайнятості. В якому навіть початківець може зробити щось дійсно суттєве та значиме. Але, як завжди, для того щоб щось почати робити потрібно зробити певний вибір, а саме: яку мову програмування вибрати? з чого почати?

Виникнення Функціонального програмування

Перші мови високого рівня були імперативні, тобто орієнтованими на послідовне виконання інструкцій, які оперують з пам'яттю. Їх теорію було закладено Джоном фон Нейманон та Аланом Тюрінгом в 30-х роках 20 століття. Математичні основи теорії ФП розробили в XX столітті, а саме 20-30 роках:

– Алонзо Черч (США) (математик та логік).

– Хаскел Каррі (Англія) (математик та логік).

В 50-х роках минулого століття, коли Джон МакКарті розробив мову програмування "*LISP*", яка практично стала першою функціональною мовою програмування. В основі "*LISP*" лежить наступний математичний апарат:

– Лямбда-числення (Алонзо Черча).

– Теорія рекурсивних функцій.

Існують мови функціонального програмування з лінивою і енергійною семантикою. Ліниві обчислення (відкладені) - це концепція в деяких мовах програмування, згідно якої обчислення відкладають до тих пір, поки не знадобиться їх результат. Найбільш відомі мови функціонального програмування це : *Haskel, Lisp, ML, Miranda, Erlang, Nemerle, F#* та інші.

На початку 80-х XX століття досить активно почали розроблятися моделі типізації, які підійдуть до функціональних мов програмування. Велика частина цих моделей включали в себе такі механізми як поліморфізм та абстракція даних. Відповідно з'являється велика кількість функціональних типізованих мов а також і їхні діалекти, а саме: *ML, Hope, Miranda* та багато інших. Але потрібно було з великої кількості наявних груп сформувати одну найбільш універсальну мову функціонально програмування, в які б було відтворено всі плюси інших мов. Як ви самі вже здогадались у них це вийшло. Мова отримала назву "*Haskell*" на честь Хаскела Каррі.

Підходи та мови програмування

Перші мови програмування, як і ЕОМ, були примітивні і орієнтувались на чисельні розрахунки. Програми були лінійними послідовностями операцій з регістрами.

З появою мов високого рівня залежність реалізації від апаратного забезпечення істотно зменшилась. Платою за це ставала поява спеціальних програм, які перетворюють інструкції умов в коди програми, та певна втрата в швидкості обчислень.

Основною відмінністю використання імперативного підходу було підвищення ефективності праці розробників за рахунок абстрагування від конкретних деталей апаратного забезпечення. Одна інструкція (оператор) мови високого рівня відповідала послідовності з декількох низькорівневих інструкцій, або команд. Виходячи з того, що програма, по суті, була набором директив, звернених до комп'ютера, такий підхід до програмування отримав назву імперативну.

Приводячи наочний приклад - мова *APL*, що трансформувалася в *BPL* і потім в *C*. Основні конструкції останнього залишаються незмінними ось вже декілька десятиліть і присутні в мові *C#*. Приклади інших імперативних мов програмування: *Basic, COBOL, Pascal.*

У 60-х роках виникає новий підхід до програмування, який до цих пір успішно конкурує з імперативним. Суть підходу полягає в тому, що програма є не набором команд, а описом дій, які необхідно здійснити. Цей підхід, як ми побачимо згодом, істотно простіше і прозоріше формалізується математичними засобами. Отже, програми простіше перевіряти на наявність помилок (тестувати), а також на відповідність заданій технічній специфікації (верифікувати).

Високий ступінь абстракції також є перевагою даного підходу. Фактично, програміст оперує не набором інструкцій, а абстрактними поняттями, які можуть бути достатньо узагальненими. Приклади декларативних мов програмування: *SML, Haskell,Ocaml, F#, Prolog*.

Однією з особливостей мов високого рівня була можливість повторного використання раніше написаних програмних блоків, що виконують ті або інші дії, за допомогою їх ідентифікації і подальшого звернення до них, наприклад по імені. Такі блоки отримали назву функцій або процедур, і програмування набуло більш впорядкованого характеру.

Функціональне та об'єктно-орієнтоване програмування

Одним з шляхів розвитку декларативного стилю програмування став функціональний підхід, що виник після створення мови LISP. Особливістю даного підходу є те, що будь-яка програма, написана на такій мові, може інтерпретуватися як функція з одним або декількома аргументами. Такий підхід дає можливість прозорого моделювання тексту програм математичними засобами, а значить, вельми цікавий з теоретичної точки зору.

Оскільки функція є природним формалізмом для мов функціонального програмування, реалізація різних аспектів програмування, пов'язаних з функціями, істотно спрощується. Зокрема, стає прозорим написання рекурсивних функцій, тобто функцій, що викликають самих себе як аргумент. Крім того, природною стає і реалізація обробки рекурсивних структур даних. Мови функціонального програмування не позбавлені недоліків. Часто до них відносять нелінійну структуру програми і відносно невисоку ефективність реалізації. Проте перший недолік достатньо суб'єктивний, а другий успішно подоланий сучасними реалізаціями.

Важливим кроком на шляху до вдосконалення мов програмування стала поява об'єктно-орієнтованого підходу до програмування (ООП) і відповідного класу мов. В рамках даного підходу програма є описом об'єктів, їх властивостей (або атрибутів), класів (або сукупностей), способів їх взаємодії і операцій над об'єктами (або методами). Безперечною перевагою даного підходу є концептуальна близь-

кість до наочної області довільної структури і призначення. Механізм спадкоємства атрибутів і методів дозволяє будувати похідні поняття на основі базових і таким чином створювати модель скільки завгодно складній наочній області із заданими властивостями.

Ще одним теоретично цікавою і практично важливою властивістю об'єктно-орієнтованого підходу є підтримка механізму обробки подій, які змінюють атрибути об'єктів і моделюють їх взаємодію в наочній області. Переміщаючись за ієрархією класів від більш загальних понять наочної області до конкретнішим (або від складніших - до простішим) і навпаки, програміст дістає можливість змінювати ступінь абстрактності або конкретності погляду на модельований їм реальний мир. Об'єкти, класи і методи можуть бути поліморфними, що робить реалізоване програмне забезпечення гнучкішим і універсальним.

Складність адекватної формалізації об'єктної теорії породжує труднощі тестування і верифікації створеного програмного забезпечення. Ймовірно, ця обставина є одним з найістотніших недоліків об'єктно-орієнтованого підходу до програмування.

Основна ідея функціонального програмування

Ідея функціонального програмування спирається на інтуїтивне уявлення про функції, як про достатньо загальний механізм постановки і аналізу складних задач. ФП ставить своєю метою надати кожній програмі просту математичну інтерпретацію. Основним в парадигмі ФП є функція. Математичні функції виражають зв'язок між параметрами (входом) і результатом (виходом) деякого процесу. Обчислення – це також процес, тому функція є відповідним і адекватним засобом опису обчислень. Цей принцип покладений в основу функціональної парадигми і функціонального стилю програмування. Функціональна програма являє собою набір визначень функцій. Функції визначаються через інші функції або рекурсивно через самих себе. Для професійної розробки програмного забезпечення на мовах функціонального програмування необхідно глибоко розуміти природу функції.

Основні принципи об'єктно-орієнтованого програмування

Об'єктно-орієнтоване програмування засноване на «трьох китах» - трьох найважливіших принципах, що додають об'єктам нові властивості. Цими принципами є інкапсуляція, спадкування та поліморфізм.

Інкапсуляція – це об'єднання в єдине ціле даних і алгоритмів обробки цих даних. В рамках ООП дані називаються полями об'єкта, а алгоритми – об'єктними методами.

Інкапсуляція дозволяє в максимальному ступені ізолювати об'єкт від зовнішнього оточення. Вона істотно підвищує надійність розроблюваних програм, тому локалізовані в об'єкті алгоритми обмінюються з програмою порівняно невеликими обсягами даних, причому кількість і тип цих даних звичайно ретельно контролюються. В результаті заміна або модифікація алгоритмів і даних, інкапсульованих в об'єкт, як правило, не тягне за собою погано простежуються наслідків для програми в цілому. Іншим важливим наслідком інкапсуляції є легкість обміну об'єктами, переносу їх з однієї програми в іншу. Можна сказати, що ООП «провокує» розробку бібліотек об'єктів, таких як *Turbo Vision*.

Спадкування є властивість об'єктів породжувати своїх нащадків. Об'єкт-нащадок автоматично успадковує від батька всі поля і методи, може доповнювати об'єкти новими полями і замінювати (перекривати) методи батька або доповнювати їх.

Принцип успадкування вирішує проблему модифікації властивостей об'єкта і додає ООП в цілому виняткову гнучкість. При роботі з об'єктами програміст звичайно підбирає об'єкт, найбільш близький за своїми властивостями для вирішення конкретного завдання, і створює одного або декількох нащадків від нього,

які «вміють» робити те, що не реалізовано в батьку.

Поліморфізм – це властивість споріднених об'єктів (тобто об'єктів, що мають одного загального батька) вирішувати схожі за змістом проблеми різними способами. В рамках ООП поведінкові властивості об'єкта визначаються набором вхідних у нього методів. Змінюючи алгоритм того чи іншого методу в нащадках об'єкта, програміст може надавати цим нащадкам відсутні у батька специфічні властивості. Для зміни методу необхідно перекрити його в нащадку, тобто оголосити в нащадку однойменний метод і реалізувати в ньому потрібні дії. В результаті в об'єкті-батьку і об'єкті-нащадку будуть діяти два однойменних методу, що мають різну алгоритмічну основу і, отже, що додають об'єктам різні властивості. Це і називається поліморфізмом об'єктів.

Так об'єктно-орієнтоване чи функціональне програмування (ООП) не може змагатися з ФП в підході до реалізації паралелізму і розподілених обчислень, тому що весь ООП ґрунтується на понятті змінності стану (зазначимо, що це особливість імперативних мов, але суть в тому, що переважна більшість з них є об'єктно-орієнтованими). Справа в об'єктних методах. Проблема виникає, коли від одного і того ж методу вимагають синхронності виконання на безлічі ядер, що в підсумку виливається в неосяжні потоки додаткового коду, а це, скажемо так, не додає простоти або гнучкості. Зазначимо що: *Java і C ++* вже товаришують з Лямбда-обчисленням. Тобто найпопулярніші мови вже починають "переодягатися" в ФП і напевно скоро до них підключаться їх менш популярні "брати". Важливо відзначити, що вам не доведеться відмовлятися від змінних станів, головна ідея ФП полягає в тому, щоб використовувати їх тільки, коли це дійсно необхідно.

Завдання та властивості ФП

В якості задач які традиційно розглядаються в програмуванні можна виділити наступні :

– Побудова математичного описання функції.

– Визначення формальної семантики мови програмування .

– Отримання залишкової процедури.

– Еквівалентна трансформація програм.

– Доказ наявності деякої властивості програми.

– Опис динамічних структур даних.

– Автоматична побудова значної частини програми по описанню структури даних, які обробляються створеною програмою.

Всі ці завдання досить легко вирішуються засобами функціонального програмування, але практично не вирішуються в імперативних мовах.

– В якості основних властивостей можна виділити наступні, а саме :

– Простота та представлення в короткому вигляді.

– Строга типізація.

– Модульність.

– Чистота, тобто відсутність будь-яких побічних ефектів.

– Відкладені обчислення.

Висновки

Перед кожним молодим програмістом постає проблема вибору, адже до цього часу є прибічники як однієї, так і іншої сторони, що всіляко намагаються зазначити плюси одного підходу та висвітлити мінуси іншого. Зазначимо що на даний момент не визначено який з підходів кращий, адже велика кількість програмістів користується ООП, але і зазначимо що в останній період ФП починає набувати все більшої популярності.

Список літератури

1. Основы функционального программирования Л.В.Городная (Новосибирск, 2004).

2. http://habrahabr.ru/post/142351/

3. Филд А., Харрисон П. Функциональное программирование. М.: Мир, 1993.

Статтю подано до редакції 22.03.2015