Common Lisp. Продолжение. CLOS, MOP

Кальянов Д.В.
Kalyanov.Dmitry@gmail.com

17 апреля 2009 года

Содержание

Объекты

Объект — это нечто, что обладает:

- Идентичностью
- Состоянием
- Поведением

Объектная система

Объектная система — часть языка программирования, определяющая семантику объектов и наделяющая их свойствами объектов.

2 классификации объектных систем:

По способу организации объектов

- Классы
- Прототипы

По способу организации поведения

- Передача сообщений
- Множественная диспетчеризация

Примеры объектных систем

	Сообщения	Диспетчеризация
Классы	Smalltalk, C++, Java, C#	CL, Dylan
Прототипы	JavaScript, Io	Slate

Содержание

Класс в CLOS

- Класс задает структуру объектов в виде именованного набора слотов.

POINT	
X	0.0
Y	0.0

COLOR-POINT	
X	0.0
Y	0.0
COLOR	rgb(1,1,1)

- Методы не принадлежат классу
- Поведение объектов задается не с помощью методов, принадледащих классу, а с помощью задания обобщенных функций, применимых к объектам класса
- Принадлежность объекта к классу используется для определения применимости методов к объекту

Определение класса

```
(defclass point ()
  ((x :initarg :x
      :accessor point-x
      :initform 0.0)
   (y :initarg :y
      :accessor point-y
      :initform 0.0)))

(defclass color-point (point)
  ((color :initarg :color
          :accessor point-color
          :initform (make-color 1.0 1.0 1.0))))
```

POINT	
X	0.0
Y	0.0

COLOR-POINT	
X	0.0
Y	0.0
COLOR	rgb(1,1,1)

Создание объектов

```
(defmethod initialize-instance :after
                                ((object point)
                                 &key x y
                                 &allow-other-keys)
  (format t "Creating point with X = ~A and Y = ~A~%"
          x y))

(make-instance 'point :x 1.0 :y 1.0)
=>
#<POINT :x 1.0 :y 1.0>
Creating point with X = 1.0 and Y = 1.0

(make-instance 'color-point :x 1.0 :color +white+)
=>
#<COLOR-POINT :x 1.0 :y 0.0 :color rgb(1,1,1)>
Creating point with X = 1.0 and Y = 1.0
```

Доступ к слотам

```
(defvar *point* (make-instance 'point))

(point-x *point*) => 0.0

(setf (point-x *point*) 4815)

(point-x *point*) => 4815

(slot-value *point* 'x) => 4815
```

Наследование

T — суперкласс всех классов (аналог класса Object)

Множественное наследование

```
(defclass point ()
  ((x ...)
   (y ...)))

(defclass colorable ()
  ((color ...)))

(defclass color-point (point colorable)
  ())
```

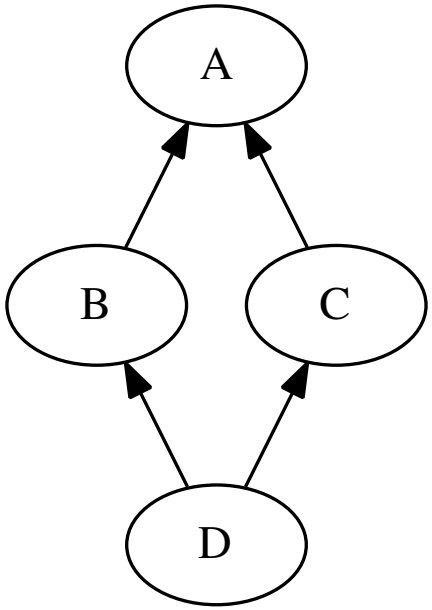
POINT	
X	0.0
Y	0.0

COLORABLE	
COLOR	rgb(1,1,1)

COLOR-POINT	
X	0.0
Y	0.0
COLOR	rgb(1,1,1)

Class precedence list

Основная проблема множественного наследования:



Для ее разрешения используется список приоритетности классов:

```
(class-precedence-list (find-class 'D))
=> (D B C A T)
```

Обобщенные функции и методы

Метод — именованная функция, и ассоциированна с классом (точнее, с упорядоченным набором классов).
Обобщенная функция — функция, «состоящая» из всех методов с одинаковым именем.

```
(defgeneric hit (object-1 object-1))

(defmethod hit ((object-1 missile) (object-2 asteroid))
  (destroy asteroid))

(defmethod hit ((object-1 missile) (ship spaceship))
  (format t "Ship␣~A␣destroyed~%" ship)
  (destroy ship))

(defmethod hit ((ship-1 spaceship) (ship-2 spaceship))
  (damage ship-1)
  (damage ship-2))
```

Комбинация методов

Комбинация методов — способ объединить действия методов

```
(defgeneric hit (object-1 object-2))

(defmethod hit (obj-1 obj-2))

(defmethod hit :after ((object-1 missile) object-2)
  (print "Destroyed") (destory object-2))

(defmethod hit :around (object-1 (ship spaceship))
  (unless (shields-holding-p ship)
    (print "Shields are penetrated") (call-next-method)))

(hit (make-missile ...) (make-spaceship :shields :off))
=>
Shields are penetrated
Destroyed
```

Диспетчеризация

Вызов обобщенной функции:

1. Определение множества методов, применимых к аргументам:
   - specializers
   - список приоритета аргументов
2. Упорядочение применимых методов по специфичности (используя Class-Precedence-List)
3. Применение методов в соответствии с комбинацией методов с учетом упорядочения и комбинирование результатов («эффективный метод»)

Такой способ вызова обобщенной функции позволяет решить проблему ромбовидной иерархии

Комбинации методов

- List — объединяет результаты применения методов в список
- Nconc — конкатенирует результаты применения методов в один список
- + — суммирует результаты применения методов
- And — вычисляет логическое И результатов применения методов
- Or — вычисляет логическое ИЛИ результатов применения методов
- Progn — применяет все применимые методы
- Можно добавлять свои комбинации методов

Пример комбинации методов

- ШахматнаяФигура, Ладья, Слон
- Слоты: X, Y, color
- Метод (possible−move−p piece x y) возвращает true, если фигура piece может быть перемещена на клетку (x,y)
- Слон ходит по диагонали, поэтому

```
(defmethod possible-move-p ((piece bishop) x y)
  (= (abs (- x (piece-x piece)))
     (abs (- y (piece-y piece)))))
```

- Ладья ходит по вертикали и горизонтали, поэтому

```
(defmethod possible-move-p ((piece rook) x y)
  (or (= x (piece-x piece))
      (= y (piece-y piece))))
```

Пример комбинации методов (продолжение)

- Ферзь ходит как ладья и как слон.
- Как задать possible−move−p для ферзя?
- Использовать комбинацию методов or!

```
(defgeneric possible-move-p (piece x y)
  (:method-combination or))
```

- Тогда для ферзя *эффективный метод* будет выглядеть как:

```
(lambda (piece x y)
  (or (= (abs (- x (piece-x piece)))
         (abs (- y (piece-y piece))))
      (or (= x (piece-x piece))
          (= y (piece-y piece)))))
```

Протокол

Протокол — набор обобщенных функций, классов и методов, совместно реализующих координированное поведение объектов.
Примеры протоколов из языка Common Lisp:

- Печать объектов. print-object
- Создание, инициализация, смена класса объектов. make-instance, initialize-instance, reinitialize-instance, update-instance-for-different-class
- Сериализация объектов. make-load-form
- Потоков ввода-вывода. stream-read-char, stream-read-line, stream-write-char, stream-position

Другие примеры:

- Взаимодействие объектов в программе физического моделирования
- Работа виджетов в программах с GUI

«Экзотика»

- CLOS позволяет переопределять классы во время работы программы
- класс объекта можно менять во время работы программы

Содержание

Метапрограммирование

Метапрограммирование — программирование языка программирования.

Метаобъектный протокол

Метаобъектный протокол — протокол, позволяющий

1. получать информацию о классах, обобщенных функциях, методах
2. вызывать функционал объектной системы — создавать объекты, создавать и вызывать методы и обобщенные функции
3. модифицировать поведение объектной системы

Метаобъектный протокол — частный случай метапрограммирования
Классам, методам, обобщенным функциям, слотам соответствуют метаобъекты, описывающие их. Метаобъекты — экземпляры метаклассов.

Интроспекция (Reflection)

Интроспекция (отражение, reflection) — часть метаобъектного протокола, позволяющая получить информацию о программе.

```
(find-class 'point) => #<STANDARD-CLASS POINT>
(class-slots *)
=>
(#<STANDARD-EFFECTIVE-SLOT-DEFINITION X>
 #<STANDARD-EFFECTIVE-SLOT-DEFINITION Y>)
(slot-definition-name (first *)) => X

(fdefinition 'point-x)
=> #<STANDARD-GENERIC-FUNCTION POINT-X (1)>

(find-method #'point-x nil (list (find-class 'point)))
=> #<STANDARD-READER-METHOD POINT-X, slot:X, (POINT)>
```

Вызов функционала CLOS

- Создание объектов: в функцию make-instance передается имя класса и аргументы:
  (**apply** #'make−instance class−name initargs)
- Доступ к слотам: в функцию slot−value передается имя слота: (slot−value object slot−name)
- Создание классов: функция ensure−class
- Создание обобщенных функций: ensure−generic−functions
- Создание методов: (make−instance 'standard−method ...), add−method
- Вызов обобщенных функций:
  (**funcall** (**fdefinition** gf−name) ...)
- Вызов методов:
  (**funcall** (standard−method−function method) ...)

Модификация поведения объектов

- Определение классов, обобщенных функций, методов: изменение class-precedence-list, добавление/удаление/модификация слотов, данные в слотах (и других метаобъектах), побочные эффекты при определении (напр., добавление методов), изменение поведения о.ф. и методов
- Создание объектов: совершение каких-то действия во время создания объектов
- Доступ к слотам: переопределение способа доступа к слотам
- Вызов обобщенных функций и методов: переопределение способа вызова функции, определения применимых методов, сортировки методов по специфичности

Сохраняется возможность компилировать эффективный код.

Пример: Elephant

Elephant — средство для прозрачной работы с реляционными СУБД как с объектными СУБД.

```
(defclass spaceship ()
  ((id :initarg :id)
   (captain :initarg :captain))
  (:metaclass persistent-metaclass)
  (:index t))

(open-store '(:CLSQL (:SQLITE "/users/me/db/sqlite.db")))

(make-instance 'spaceship :id "NCC-1701"
                          :captain "James T. Kirk")

(get-instances-by-value 'spaceship 'id "NCC-1701")
=>
(#<SPACESHIP NCC-1701 commanded by James T. Kirk>)
```

Интеграция с другими объектными системами

Использование MOP иногда позволяет добиться тесной интеграции разных объектных систем. Например, GObject:

```
(defclass gtk-label ()
  ((angle :allocation :gobject
          :g-property-name "angle"
          :g-property-type "gdouble"
          :accessor gtk-label-angle)
   (label :allocation :gobject
          :g-property-name "label"
          :g-property-type "gchararray"
          :accessor gtk-label-label))
  (:metaclass gobject-class)
  (:g-type-name . "GtkLabel"))
```

Что дает CLOS

- Более лучшая поддержка модульности и повторного использования кода
  - Множественное наследование
  - Комбинации методов
  - Протоколы
  - Множественная диспетчеризация
- Расширяемость

Паттерны проектирования vs свойства языка

Есть мнение, что т.н. «паттерны проектирования» есть способ обхождения отсутствия какой-то возможности в языке. Отстутствие явной поддержки тех или иных паттернов является нарушением принципа DRY — Do Not Repeat Yourself.

- Visitor vs множественная диспетчеризация
- Command vs функции как значения
- Strategy vs протоколы
- Factory vs make-instance
- Interpreter vs макросы
- Compiler vs compile
- Template method vs обобщенные функции

Содержание

Those who cannot remember the past are condemned to repeat it

— George Santayana (1863–1952), философ, поэт