Шаблоны ООП, регулярные выражения

Николай Вяххи

Санкт-Петербургский Государственный Университет

16 апреля 2010 г.

Шаблоны

Существует два термина, означающих три разные вещи:

- Template (шаблон) — средство языка, предназначенное для кодирования обобщенных алгоритмов (например в C++, в Java похожие конструкции называются Generics).
- Pattern (образец):
  - Design Pattern (шаблон проектирования) — многократно применяемая архитектурная конструкция, предоставляющая решение общей проблемы проектирования.
  - Pattern Matching (сопоставление с образцом) — метод анализа списков или других структур данных на наличие в них заданных образцов.

Design Patterns

Каждый шаблон имеет:

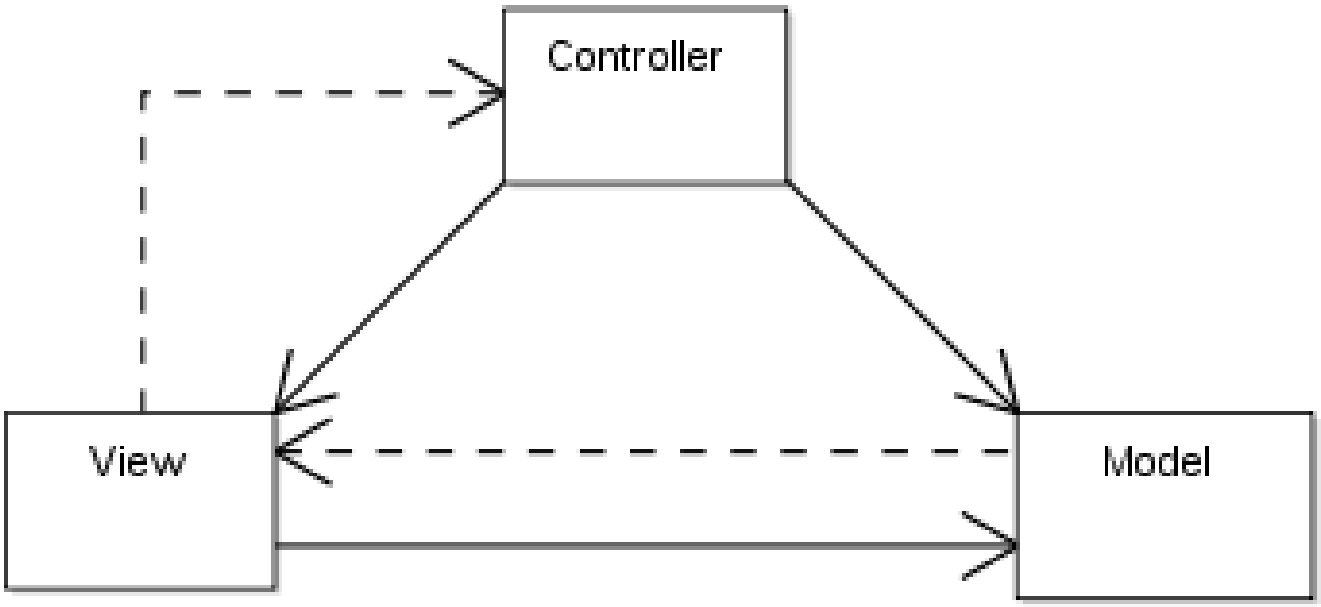
- Имя — как на него ссылаются в компании умных людей.
- Задачу — когда следует применять паттерн.
- Решение — описание элементов дизайна, отношений между ними, функций каждого элемента.
- Результаты — следствия применения паттерна и разного рода компромиссы.

Design Patterns

Шаблоны бывают:

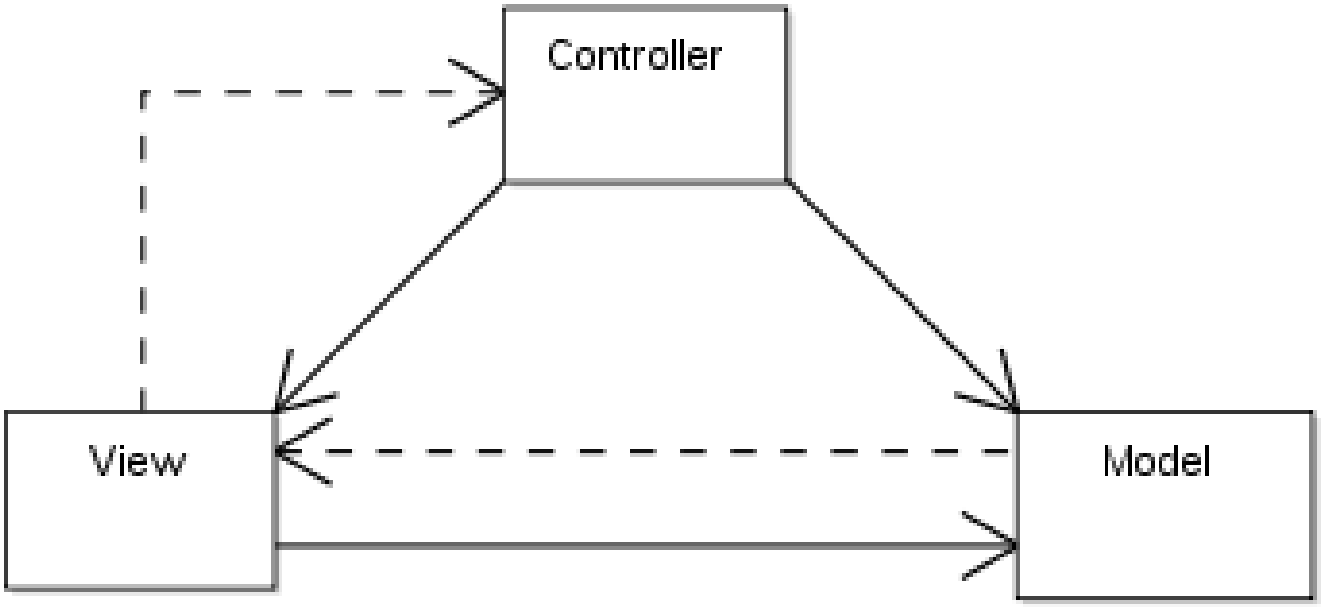
- MVC — Model-View-Controller.
- Порождающие (Creational) — что-то создают.
- Структурные (Structural) — отражают структу.
- Поведенческие (Behavioral) — моделируют поведение.

MVC



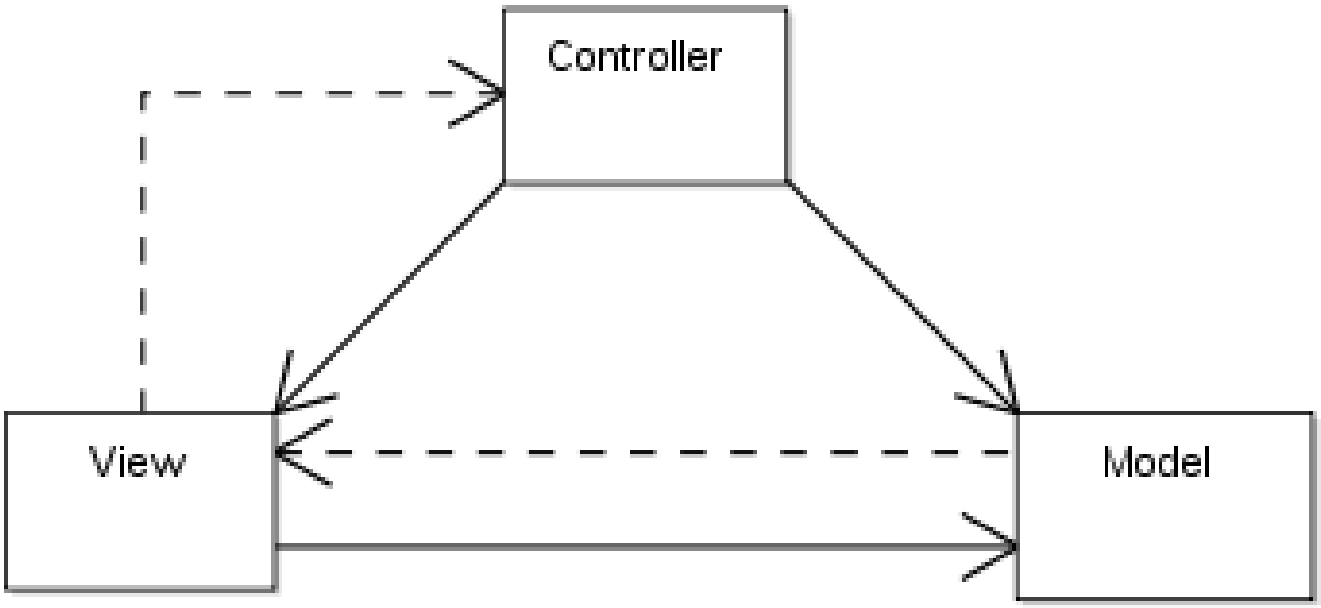
- Model (модель) – предоставляет данные (обычно для View), а также реагирует на запросы (обычно от контроллера), изменяя свое состояние.

MVC



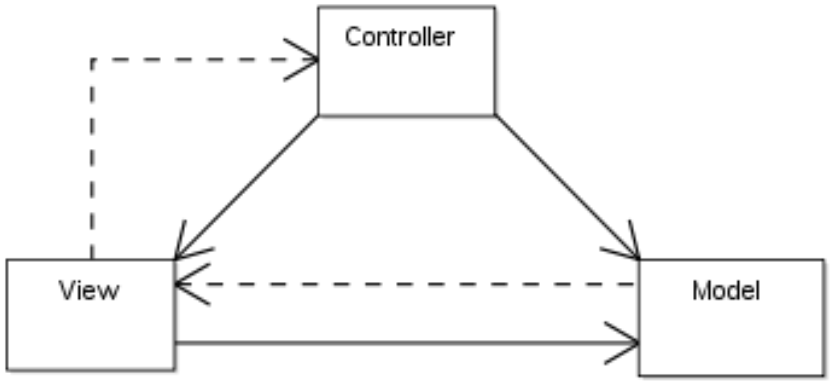
- View (представление) – отвечает за отображение информации (пользовательский интерфейс).

MVC



- Controller (контроллер) – интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.

MVC



Как представление, так и поведение зависят от модели!

Однако модель не зависит ни от представления, ни от поведения.

Это позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Abstract Factory (Фабрика)

```
1  class GUI(object):
2      def make_button(self):
3          pass
4      def make_window(self):
5          pass
6
7  class MacGUI(GUI):
8      def make_button(self):
9          ...
10         return ...
11     def make_window(self):
12         ...
13         return ...
```

Abstract Factory (Фабрика)

```
1  def create_chart(gui):
2      assert isinstance(gui, GUI)
3      ...
4      gui.make_button()
5      ...
6      gui.make_window()
7      ...
8
9  gui = MacGUI() # or WinGUI()
10 ...
11 chart1 = create_chart(gui)
12 chart2 = create_chart(gui)
```

Singleton (Одиночка)

```
1  class Singleton(object):
2      __instance = None
3      def __new__(cls, *args, **kwargs):
4          if not cls.__instance:
5              cls.__instance = super(Singleton, cls).__new__(
6                                  cls, *args, **kwargs)
7          return cls.__instance
8
9  class C(Singleton):
10     pass
11
12 [a, b] = [ C(), C() ]
13 print id(a), id(b) # 34807144 34807144
```

Adapter, Wrapper (Адаптер, Обертка)

```
1  class Adaptee(object):
2      def __init__(self, *args, **kw):
3          pass
4      def specific_request(self):
5          return ' Adaptee '
6
7  class Adapter(object):
8      def __init__(self, adaptee):
9          self.adaptee = adaptee
10     def request(self):
11         return self.adaptee.specific_request()
12
13 client = Adapter(Adaptee())
14 print client.request()
```

Command (Команда)

```
1  class Command(object):
2      def execute(self):
3          pass
4
5  class OpenCommand(Command):
6      def __init__(self, document):
7          self.__document = document
8      def execute(self):
9          ...
10
11 history = [OpenCommand('my.doc'), PasteCommand(
12         'my.doc', text, pos), SaveCommand('my.doc')]
13 for cmd in history:
14     cmd.execute()
```

Observer (Наблюдатель)

```
1  class Subject(object):
2      def __init__(self):
3          self.__observers = set()
4      def attach(self, observer):
5          self.__observers.add(observer)
6      def notify(self):
7          for o in self.__observers:
8              o.update(self)
9
10 class Observer(object):
11     def update(self, subject):
12         print subject, ' notified me! '
13
14 s = Subject()
15 s.attach(Observer())
16 s.notify()
```

Полезные ссылки

Gamma, Helm, Johnson, Vlissides: Приемы ООП Паттерны проектирования
(Design Patterns – Elements of Reusable Object-Oriented Software)
http://rutracker.org/forum/viewtopic.php?t=702982

Patterns in Python
http://www.suttoncourtenay.org.uk/duncan/accu/pythonpatterns.html

Шаблон проектирования – Википедия
http://ru.wikipedia.org/wiki/Шаблон_проектирования
http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

Regular Expressions

Регулярные выражения - формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

Regular Expressions (regexp)

```
1  import re
2
3  p = re.compile( ' ab* ' , re.IGNORECASE)
4
5  m = p.match( ' ABBBCDAB ' )
6  if m:
7      print  ' Match found: ' , m.group(), # ABBB
8      print  ' in positions from ' , m.start(),  ' to ' , m.end() #
9  else:
10     print  ' No match '
11
12 for m in p.finditer( ' ABBBCDAB ' ):
13     print m.group(), m.span(), # ABBB (0, 4) AB (6, 8)
14
15 print p.findall( ' ABBBCDAB ' ) # ['ABBB', 'AB']
```

Метасимволы

. (точка) – любой символ.

Символьные классы:

- $[abc]$ – либо a, либо b, либо c
- $[a - zA - Z]$ – буква латинского алфавита
- $[\hat{}0 - 9]$ – не цифра

Позиция внутри строки:

- $\hat{}$ – начало строки
- \$ – конец строки

Метасимволы

Квантификация (поиск последовательностей):

- $a\{10\}$ – ровно 10 символов a
- $a\{10, 15\}$ – от 10 до 15 символов a включительно
- $a\{10, \}$ – не менее 10 символов a
- $a\{, 15\}$ – не более 15 символов a
- $a*$ – ноль или более символов a (т.е. $a\{0, \}$)
- $a+$ – один или более символов a (т.е. $a\{1, \}$)
- $a?$ – ноль или один символ a (т.е. $a\{0, 1\}$)

Жадная, ленивая и ревнивая квантификация:

- $a*$ – как можно **больше** a, лишь бы сошлось
- $a*?$ – как можно **меньше** a, лишь бы сошлось
- $a*+$ – как можно **больше** a

Группировка

- **(тр[ау]м-?)*** найдет последовательность вида **трам-трам-трумтрам-трум-трамтрум**
- **(та|ту)-\1** найдет строку **та-та** или **ту-ту**, но пропустит строку **та-ту**
- Можно использовать m.group(1) чтобы вывести 1-ую группу из найденного выражения
- Нулевая группа – все выражение, а m.group() = m.group(0)

Некоторые сокращения

- $\backslash w$ равно $[A-Za-z0-9\_]$
- $\backslash W$ равно $[\hat{}\backslash w]$
- $\backslash d$ равно $[0-9]$
- $\backslash D$ равно $[\hat{}\backslash d]$
- $\backslash s$ равно $[\ \backslash t\backslash r\backslash n\backslash v\backslash f]$
- $\backslash S$ равно $[\hat{}\backslash s]$
- шаблон $\backslash\backslash$ найдется в тексте $\backslash$
- шаблон $\backslash *$ найдется в тексте $*$

Полезные ссылки

Regular Expression HOWTO
http://www.amk.ca/python/howto/regex/

Python docs: re – Regular expression operations
http://docs.python.org/library/re.html

Регулярные выражения – Википедия
http://ru.wikipedia.org/wiki/Регулярные_выражения
http://en.wikipedia.org/wiki/Regular_expression