Декларативно програмиране в процедурните езици

Ламбда изрази

Владислав Ненчев

Софийски Университет “Св. Климент Охридски”
Факултет по Математика и Информатика
Катедра по Математическа Логика и Приложенията ѝ

2 Декември 2016

Ламбда изрази - запис

```
int f(int i, int j, int n)
{
  return (g(i) + h(j)) / n;
}
```

$$\lambda i, j, n.\ (g(i) + h(j))/n$$

Ламбда изрази - запис

```
int f(int i, int j, int n)
{
  return (g(i) + h(j)) / n;
}
```

$$\lambda i, j, n.\ (g(i) + h(j))/n$$

Позволява анонимни функции (употребяване по веднъж или само в локална област). Позволява функционално програмиране (функции от по-висок ред). Позволява декларативен запис (извличане на свойства на функциите).

Ламбда изрази - запис

```
int f(int i, int j, int n)
{
  return (g(i) + h(j)) / n;
}
```

$$\lambda i, j, n.\ (g(i) + h(j))/n$$

Позволява анонимни функции (употребяване по веднъж или само в локална област). Позволява функционално програмиране (функции от по-висок ред). Позволява декларативен запис (извличане на свойства на функциите).

Не е подходящо за рекурсия, или многократно използване, или използване в далечни области, или за публични методи.

Основни принципи на функционалното програмиране

Същински функции (без странични ефекти):

- ясна и енкапсулирана логика;
- по-добър контрол върху ефектите/изчисленията (по-голяма сигурност);
- по-лесна проверка;
- автоматична верификация;
- автоматично извличане/разпознаване на свойства.

Основни принципи на функционалното програмиране

Същински функции (без странични ефекти):

- ясна и енкапсулирана логика;
- по-добър контрол върху ефектите/изчисленията (по-голяма сигурност);
- по-лесна проверка;
- автоматична верификация;
- автоматично извличане/разпознаване на свойства.

Функции от по-висок ред:

- по-гъвкав, систематизиран и енкапсулиран код;
- функциите са данни.

Ламбда изрази в C#

```
Height = children.Select(child => child.Height + 1).
  DefaultIfEmpty().Max();
```

```
bool AreConnected(GraphVertex x, GraphVertex y) =>
  DFSTraversal(x).Contains(y);
```

Лямбда изрази в C#

```
Height = children.Select(child => child.Height + 1).
  DefaultIfEmpty().Max();
```

```
bool AreConnected(GraphVertex x, GraphVertex y) =>
  DFSTraversal(x).Contains(y);
```

Делегати за работа с функции:

- Func<T, TResult>, ..., Func<T1, ..., T16, TResult>;
- Action<T>, ..., Action<T1, ..., T16>;
- Predicate<T>.

Използване на Func, Action и Predicate

```
public static class IEnumerableExtensions
{
  public static void ForEach<T>(
    this IEnumerable<T> iEnumerable, Action<T> action)
  {
    foreach(T item in iEnumerable)  action(item);
  }
}

...

  void printInfo(int number) ...

  void processNumbers(IEnumerable<T> numbers)
  {
    Predicate<int> condition = n => n > 20;
    Action<int> action = printInfo;
    numbers.Where(condition).forEach(action);
  }
```

Лямбда изрази в Java

Interface-и за работа с функции:

- Function<T, R>, Function.compose, Function.andThen, Function.apply, Function. identity ;
- BiFunction<T,U,R>, BiFunction.andThen, BiFunction.apply;
- Consumer<T>, Function.andThen, Function.accept;
- BiConsumer<T,U>, BiConsumer.andThen, BiConsumer.accept;
- Supplier <T>, Supplier .get;
- Predicate<T>, Predicate .and, Predicate .negate, Predicate .or, Predicate . test ;
- BiPredicate<T,U>, BiPredicate .and, BiPredicate .negate, BiPredicate .or, BiPredicate . test .

Намиране на най-честото разстояние между редове

```
class LineData
{
  public String LineText;
  public int LineHeight;
}

class Page
{
  ArrayList<LineData> data;

  ArrayList<Integer> LineSpacingData()
  {
    return IntStream.range(0, data.size()-1).boxed().
      map(index -> data.get(index).LineHeight -
        data.get(index + 1).LineHeight).
          collect(Collectors.
            toCollection(ArrayList::new));
  }
}
```

Намиране на най-честото разстояние между редове (продължение)

```
Map<Integer, Long> lineSpacingCounts =
  document.stream().flatMap(page ->
    page.LineSpacingData().stream()).
      collect(Collectors.groupingBy(
        Function.identity(), Collectors.counting()));

int mostCommonLineSpacing =
  lineSpacingCounts.keySet().stream().
    reduce((key1, key2) ->
      lineSpacingCounts.get(key1) >
      lineSpacingCounts.get(key2) ? key1 : key2).
        orElse(-1);
```

Лямбда изрази в JavaScript

```
wu(/(25|31)\.12\.\d{2}(\d{2})/g.AllMatches(text)).
  filter(match => parseInt(match[2]) % 4 == 0).
    forEach(match => console.log(match[0]));
```

```
function MultiIntersection()
{
  ...
  return
    new Set([...minimalSizedSet].
      filter(element =>
        [...arguments].every(set =>
          set.has(element))));
};
```

Ламбда изрази в JavaScript

```
wu(/(25|31)\.12\.\d{2}(\d{2})/g.AllMatches(text)).
  filter(match => parseInt(match[2]) % 4 == 0).
    forEach(match => console.log(match[0]));
```

```
function MultiIntersection()
{
  ...
  return
    new Set([...minimalSizedSet].
      filter(element =>
        [...arguments].every(set =>
          set.has(element))));
};
```

Имената на функциите се използват директно като данни.
Всички функции имат методи call и apply (става и с употреба на () след функционална променлива).

Частично използване/прилагане на функции

```
var partial = function() {
    var values = [...arguments];
    var functionName = values.shift();
    function partialExecution()
    {
      var newValues = [...arguments];
      return functionName.apply(this,
        values.concat(newValues));
    }
    return partialExecution;
  }
```

Частично използване/прилагане на функции

```
var partial = function() {
    var values = [...arguments];
    var functionName = values.shift();
    function partialExecution()
    {
      var newValues = [...arguments];
      return functionName.apply(this,
        values.concat(newValues));
    }
    return partialExecution;
  }
```

```
var wrap = function(tag, text) {
    return "<" + tag + ">" + text + "</" + tag + ">";
  }

var wrapParagraph = partial(wrap, "p");
```

Композиция на функции

```
var compose = function() {
    var values = arguments;
    var start = values.length - 1;
    function composition()
    {
      var result = values[start].apply(this,
          arguments);
      start = start - 1;
      while (start >= 0)
      {
        result = values[start].call(this, result);
        start = start - 1;
      }
      return result;
    }
    return composition;
  }
```

Композиция на функции (продължение)

```
var wrap =
  function(tag, text)
  {
    return "<" + tag + ">" + text + "</" + tag + ">";
  }
var replaceNewLines =
  function(replacement, text)
  {
    return text.replace(/\n/g, replacement);
  }

var addBreaks = partial(replaceNewLines, "<br/>\n");
var wrapParagraph = partial(wrap, "p");
var wrapBlockquote = partial(wrap, "blockquote");

var prepareForHTML =
  compose(wrapBlockquote, wrapParagraph, addBreaks);
```

Лямбда изрази в Python

```
minimal4DigitPalindrome = First(PrimeNumbers,
  lambda prime: prime > 999 and
    Reverse(str(prime)) == str(prime))
```

```
def LessThan(sequence, limit):
  return takewhile(
    lambda number: number <= limit, sequence())
```

Лямбда изрази в Python

```
minimal4DigitPalindrome = First(PrimeNumbers,
  lambda prime: prime > 999 and
    Reverse(str(prime)) == str(prime))
```

```
def LessThan(sequence, limit):
  return takewhile(
    lambda number: number <= limit, sequence())
```

Имената на функциите се използват директно като данни.
Полезни функции и оператори от модули functools и operator:
functools.partial, operator.add, operator.mul, operator.pow.

Ефективност на ламбда изразите

```
def EratosthenesLambda():
  generated = []
  for number in count(2):
    if all(map(lambda lesser:
      number % lesser > 0, generated)):
        generated.append(number)
        yield number

def EratosthenesNonLambda():
  generated = []
  for number in count(2):
    isPrime = True
    for lesser in generated:
      if number % lesser == 0:
        isPrime = False
        break
    if isPrime:
      generated.append(number)
      yield number
```