Декларативно програмиране в процедурните езици

Метод на поточната линия

Владислав Ненчев

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

11 Ноември 2016

Метод на поточната линия

Основна идея: Разделяне на изчислението на *линейна* последователнот от действия. Резултатът от всяко действие се използва за входни данни на следващото.

Метод на поточната линия

Основна идея: Разделяне на изчислението на *линейна* последователнот от действия. Резултатът от всяко действие се използва за входни данни на следващото.

Изисква:

- логиката на изчислението да е без разклонения;
- независима обработка на обектите.

Метод на поточната линия

Основна идея: Разделяне на изчислението на *линейна* последователнот от действия. Резултатът от всяко действие се използва за входни данни на следващото.

Изисква:

- логиката на изчислението да е без разклонения;
- независима обработка на обектите.

Позволява:

- прост и ясен запис на алгоритъма;
- оптимизиране чрез паралелизация.

Метод на поточната линия

Основна идея: Разделяне на изчислението на *линейна* последователнот от действия. Резултатът от всяко действие се използва за входни данни на следващото.

Изисква:

- логиката на изчислението да е без разклонения;
- независима обработка на обектите.

Позволява:

- прост и ясен запис на алгоритъма;
- оптимизиране чрез паралелизация.

Не е подходящо за: рекурсия и backtracking.

Примери

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

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

```
current = outgoing.get(current.getAsInt()).
            stream().map(edge -> edge.To).
              mapToInt(Integer::intValue).
                max();
```

Примери (продължение)

```
String ReplaceIdsPipeline(String ids,
  ArrayList<Party> parties)
{
  return
    Stream.of(ids.split(",")).
    map(part -> Integer.parseInt(part)).
    map(id -> parties.stream().filter(party ->
        party.PartyId() == id).findFirst().get()).
    filter(party -> party.Type() == PartyType.Person).
    sorted(new PartyComparator()).
    map(person -> person.PartyName() + " - " +
        person.PartyDescription()).
    collect(Collectors.joining("\n"));
}
```

Примери (продължение)

```
String ReplaceIdsPipeline(String ids,
  ArrayList<Party> parties)
{
  return
    Stream.of(ids.split(",")).
    map(part -> Integer.parseInt(part)).
    map(id -> Party.GetPartyById(parties, id)).
    filter(party -> party.IsPerson()).
    sorted(new PartyComparator()).
    map(person -> person.GetDisplayText()).
    collect(Collectors.joining("\n"));
}
```

Разклонения в логиката I

Извличат се не само Person обекти, ами всички *валидни* Party обекти. Person обектите са валидни ако описанието им съдържа конкретен текст, а Company обектите са валидни ако името им завършва по определен начин.

Разклонения в логиката I

Извличат се не само Person обекти, ами всички *валидни* Party обекти. Person обектите са валидни ако описанието им съдържа конкретен текст, а Company обектите са валидни ако името им завършва по определен начин.

```
return
  Stream.of(ids.split(",")).
  map(part -> Integer.parseInt(part)).
  map(id -> Party.GetPartyById(parties, id)).
  filter(party -> party.IsValid()).
  sorted(new PartyComparator()).
  map(person -> person.GetDisplayText()).
  collect(Collectors.joining("\n"));
```

Разклонения в логиката II

Извличат се всички Party обекти и след това ако обекта е Person се изписва на екран името му с главни букви, а ако е Company се изписва броя на думите в описанието му.

Разклонения в логиката II

Извличат се всички Party обекти и след това ако обекта е Person се изписва на екран името му с главни букви, а ако е Company се изписва броя на думите в описанието му.

```
return
  Stream.of(ids.split(",")).
  map(part -> Integer.parseInt(part)).
  map(id -> Party.GetPartyById(parties, id)).
  forEach(party -> party.PipelineProcess());
```

Разклонения в логиката II

Извличат се всички Party обекти и след това ако обекта е Person се изписва на екран името му с главни букви, а ако е Company се изписва броя на думите в описанието му.

```
return
  Stream.of(ids.split(",")).
  map(part -> Integer.parseInt(part)).
  map(id -> Party.GetPartyById(parties, id)).
  forEach(party -> party.PipelineProcess());
```

Извличат се всички Party обекти, като за Person обектите се намират само валидните, сортират се и имената им се изписват с главни букви, а за Company обектите се изписва броя на думите в описанието, но само за тези, които съдържат адрес.

Курсова задача II - примери

Задача:
Да се реализира метод, който проверява дали подадена итерация/поток от обекти отговаря на "функционален" регулярен израз. "Функционалните" регулярни изрази са от вида $(f_{i_1}\&f_{i_2}\&\ \ldots\ \&f_{i_k})\{n,m\}?$, където f_i са вече реализирани статични булеви функции, които могат да се прилагат за конкретен обект, & обозначава едновременност, а $\{n,m\}?$ е стандартното означение от регулярните изрази.
Вход: Списък от обекти.
Вече реализирани методи: static bool f_1(object), static bool f_2(object), ... static bool f_n(object).
Изход: True/False.

Курсова задача II - примери (продължение)

Задача:
Да се реализира итератор, който обхожда подаден граф с числа по възлите, чрез търсене в лъч. Пироритета на обхождане на възлите се определя от числата в тях (по-големите се обхождат първи). Широчината на лъча се подава като параметър.
Вход: Граф с числа по възлите и широчина на лъча.
Изход (на екран): Итерира възлите, през които се минава по време на обхождането.

Задача:
Да се реализира итератор, който изброява всички подграфи на даден граф.
Вход: Граф.
Изход (на екран): Итерира подграфите.

Курсова задача II - примери (продължение)

Задача:
Имаме пространство (екран) с размер NxM (пиксела). В него са разположени няколко квадрата (с дадени координати на центъра и размер на страната). Да се реализира итератор, който изброява възможните позиции на нов квадрат (размера на страната му се подава като параметър) така, че той да не се пресича с вече съществуващи квадрати и да е възможно най-близко до центъра на пространството, но не по-близко от дадено ограничение (минимално разстояние от центъра). Използват се само целочислени координати.
Вход: N, M, списък от съществуващите квадрати, минимално разстояние от центъра и размер на страната на новия квадрат.
Изход (на екран): Итерира възможните позиции на центъра на новия квадрат.

Курсова задача II - примери (продължение)

Задача:
Да се реализира итератор, който изброява матриците от редица от генетично получени квадратни матрици с рационални числа. За всяка матрица е определено число, което представлява нейния живот (в брой итерации на итератора). Редицата започва с две матрици с еднакъв размер, а всяка следваща матрица се получва, като на случаен принцип се изберат две все още живи (по време на текущата итерация) матрици и се образува нова метрица от средно-аритметичното им. Живота на новата матрица е произволно число от 0 до 5. Една матрица живее в продължение на толкова итерации, колкото е размера на живота ѝ.
Вход: Двете начални матрици и техните животи (от 2 до 5).
Изход (на екран): Итерира матриците от редицата.

Курсова задача II - примери (продължение)

Задача:
Да се реализира итератор на елементарните цикли в даден граф с тежести по ребрата. Итератора да се използва за намиране на сумите от ребрата по циклите, които са по-дълги от половината от броя на върховете на графа.
Вход: Граф с тежести по ребрата.
Изход (на екран): Итерира сумите на дългите цикли.

Задача:
Дадени са вече реализираните итератори I_1, I_2, ... I_n на цели числа. Да се реализира итератор, който изброява следните стойности: $min(I_1[0], I_2[0] \ldots I_n[0])$, $max(I_1[1], I_2[1] \ldots I_n[1])$, $min(I_1[2], I_2[2] \ldots I_n[2])$, ... и т.н. Ако итератор завърши преди другите, се счита че той генерира нули.
Вече реализирани методи: Итератори/генератори I_1, I_2, ... I_n.
Изход (на екран): Итерира първите 20 числа от новия итератор.

Курсова задача II - примери (продължение)

Задача:
Да се реализира итератор на страниците (с размер N реда) на даден файл и чрез него да се намерят тези страници, в които броя на символите е по-голям или равен от 50N и за всяка от тях да се изведат броя на символите и броя на думите.
Вход: Файл и число N.
Изход (на екран): Итерира брой символи и думи.

Задача:
Даден е монотонен оператор Φ (оператор Φ, който преобразува множества и за всяко множество X имаме, че $X \subseteq \Phi(X)$). Да се намери момента на насищане на прилагането на Φ към подадено множество X.
Вход: Множество X.
Вече реализирани методи: Оператор Φ.
Изход: Резултата $\Phi^n(X)$ при който се постига насищането.