Proudy (Stream) v Java 8

Tomáš Pitner, upravil Marek Šabo

Kolekce typu Stream

- proud (java.util.stream.Stream) je typ zavedený od Java 8
- neplést se vstupně/výstupními proudy (java.io.InputStream/OutputStream)
- jde jednoduše řečeno o homogenní lineární strukturu prvků, tedy obdobně jako např. seznam nebo kolekce
- hodí se pro snadné *řetězení více operací* nebo tam,
- kde se jedná o zpracování dat, která v době spuštění nemusejí dosud (nebo vůbec nikdy) celá aktuálně existovat, nýbrž jsou dle potřeby generována
- jedná se o určitý typ "lazy collections" na rozdíl od klasických seznamů apod., které všechny prvky aktuálně obsahují (odkazují se na ně) a kde to není tak, že by se prvek objevil, až teprve ho budeme potřebovat.

Kdy proud použít?

Bez splnění aspoň jedné následující charakteristiky nemají proudy zásadnější smysl:

- využíváme řetězení operací
- využíváme možnost paralelizace těchto operací (hromadné provedení nad prvky proudu), což je jediná cesta, jak na moderních vícejádrových strojích zvýšit výkon
- pomůže to pro zpřehlednění programu, jeho zhutnění

Předpoklady využití

Proudy nelze prakticky příliš využít bez současného zvládnutí *funkcionálních prvků* v Java 8:

- *odkazy na funkce (metody)* a
- *lambda výrazy*.

Reálně bude ještě delší dobu trvat, než se začnou masověji objevovat v kódu:

- neznalost u vývojářů
- setrvačnost v běhových prostředích (možno až od Java 8 a ta skoro nikde není nasazena)
- nezralost API (např. výkonnostní problémy u paralelizace v cloudových, ale i jiných prostředích)

Vytvoření Stream

- Nejčastěji a nejjednodušeji z prvků pole nebo kolekce, tzn. např.
  - List<String> names = ...; names.stream() nebo
  - String[] names = ...; Stream.of(names)

- Proudy pro následné paralelní zpracování lehce obdobně vytvoříme pomocí např. names.parallelStream()
- tento postup tedy nevede k vytvoření proudu, kde se prvky "vyrábějí" dle potřeby, ale kde již od počátku všechny jsou.

Příklad použití Stream

Dále lze proud použít pro řetězení operací nad jeho prvky:

```
List<String> names = ...
names.stream().map(String::toUpperCase).forEach(System.out::println);
```

1. Nejdříve vytvoří ze seznamu proud
2. pak každý řetězec převede pomocí toUpperCase (průběžná operace)
3. na závěr každý takto převedený řetězec vypíše (terminální/koncová operace)

Zhruba odpovídá sekvenční iteraci:

```
List<String> names = ...
for(String name: names) {
   System.out.println(name.toUpperCase());
}
```

Stream s lambda výrazy

```
List<Integer> integerList = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
integerList.parallelStream().forEach(i -> System.out.print(i + " "));
```

nebo s více stupni řetězení

```
List<Integer> integerList = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
integerList
   .parallelStream()
   .filter(i -> isOdd(i))
   .forEach(i -> System.out.print(i + " "));
```

Dokumentace

- Benjamin Winterberg: Java 8 Stream Tutorial
- Amit Phaltankar: Understanding Java 8 Streams API