Dynamické datové struktury (kontejnery)

Obsah

Kontejnery, iterátory, kolekce

- Kontejnery jako základní dynamické struktury v Javě
- Kolekce, iterátory (Collection, Iterator)
- Seznamy (rozhraní List, třídy ArrayList, LinkedList)
- Množiny (rozhraní Set, třída HashSet), uspořádané množiny (rozhraní SortedSet, třída TreeSet), rozhraní Comparable, Comparator
- Mapy (rozhraní Map, třída HashMap), uspořádané mapy (rozhraní SortedMap, třída TreeMap)
- Klasické netypové vs. nové typové kontejnery - generické datové typy
- Iterace cyklem foreach

- Starší typy kontejnerů (Vector, Stack, Hashtable)

Kontejnery

Kontejnery (containers) v Javě

- slouží k ukládání objektů (ne hodnot primitivních typů!)
- v Javě byly koncipovány jako *beztypové* - to už ale ve verzi 1.5 neplatí!
- v Javě 1.5 mají kolekce *typové parametry* (vyznačené ve špičatých závorkách (např. List<Person>), jimiž určujeme, jaké položky se do kolekce smějí dostat

Většinou se používají kontejnery hotové, vestavěné, tj. ty, jež jsou součastí Java Core API:

- vestavěné kontejnerové třídy jsou definovány v balíku `java.util` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=java.util]
- je možné vytvořit si vlastní implementace, obvykle ale zachovávající/implementující „standardní“ rozhraní

K čemu slouží?

- jsou dynamickými alternativami k poli a mají daleko širší použití
- k uchování proměnného počtu objektů -
- počet prvků se v průběhu existence kontejneru může měnit
- oproti polím nabízejí časově efektivnější algoritmy přístupu k prvkům

Základní kategorie kontejnerů

- seznam (*List*) - lineární struktura, každý prvek má svůj číselný index (pozici)
- množina (*Set*) - struktura bez uspořádání, rychlé dotazování na přítomnost prvku
- asociativní pole, mapa (*Map*) - struktura uchovávající dvojice klíč->hodnota, rychlý přístup přes klíč

Kontejnery - rozhraní, nepovinné metody

- Funkcionalita vestavěných kontejnerů je obvykle předepsána výhradně *rozhraním*, jenž implementu-

jí.

- Rozhraní však připouštějí, že některé metody jsou *nepovinné*, třídy jej nemusí implementovat!
- V praxi se totiž někdy nehodí implementovat jak čtecí, tak i zápisové operace - některé kontejnery jsou „read-only“

Kontejnery - souběžný přístup, výjimky

- Moderní kontejnery jsou *nesynchronizované*, nepřipouštějí souběžný přístup z více vláken.
- Standardní, nesynchronizovaný, kontejner lze však „zabalit“ synchronizovanou obálkou.
- Při práci s kontejnery může vzniknout řada *výjimek*, např. *IllegalStateException* apod.
- Většina má charakter výjimek *běhových*, není povinností je odchytávat - pokud věříme, že nevzniknout.

Iterátory

Iterátory jsou prostředkem, jak "chodit" po prvcích kolekce buďto

- v neurčeném pořadí nebo
- v uspořádání (u uspořádaných kolekcí)

Každý iterátor musí implementovat velmi jednoduché rozhraní *Iterator* se třemi metodami:

- *boolean hasNext()*
- *E next()*
- *void remove()*

Kolekce

- jsou kontejnery implementující rozhraní *Collection* - API doc k rozhr. *Collection* [http://java.sun.com/j2se/1.5/docs/api/java/util/Collection.html]
- Rozhraní kolekce popisuje velmi obecný kontejner, disponující operacemi: *přidávání, rušení prvku, získání iterátoru, zjišťování prázdnosti* atd.
- Mezi kolekce patří mimo *Map*y všechny ostatní vestavěné kontejnery - *List*, *Set*
- Prvky kolekce nemusí mít svou pozici danou indexem - viz např. *Set*

Seznamy

Seznamy

- lineární struktury
- implementují rozhraní *List* (rozšíření *Collection*) API doc k rozhr. *List* [http://java.sun.com/j2se/1.5/docs/api/java/util/List.html]
- prvky lze adresovat indexem (typu int)
- poskytují možnost získat dopředný i zpětný *iterátor*
- lze pracovat i s *podseznamy*

Množiny

Množiny

Množiny

- jsou struktury standardně bez uspořádání prvků (ale existují i uspořádané, viz dále)
- implementují rozhraní *Set* (což je rozšíření *Collection*)

Cílem množin je mít možnost rychle (se složitostí O(log(n))) provádět atomické operace:

- vkládání prvku (*add*)
- odebírání prvku (*remove*)
- dotaz na přítomnost prvku (*contains*)
- lze testovat i relaci *„je podmnožinou“*

Standardní implementace množiny:

- *hašovací tabulka* (*HashSet*) nebo
- *vyhledávací strom* (černobílý strom, Red-Black Tree - *TreeSet*)

Uspořádané množiny

Uspořádané množiny:

- Implementují rozhraní *SortedSet* - API doc k rozhraní *SortedSet* [http://java.sun.com/j2se/1.5/docs/api/java/util/SortedSet.html]
- Jednotlivé prvky lze tedy iterátorem procházet v přesně definovaném pořadí - uspořádání podle *hodnot prvků*.
- Existuje vestavěná impl. *TreeSet* - černobílé stromy (Red-Black Trees) API doc ke třídě *TreeSet* [http://java.sun.com/j2se/1.5/docs/api/java/util/TreeSet.html]

Uspořádání je dáno buďto:

- standardním chováním metody *compareTo* vkládaných objektů - pokud implementují rozhraní *Comparable*
- nebo je možné uspořádání definovat pomocí tzv. *komparátoru* (objektu impl. rozhraní *Comparator*) poskytnutých při vytvoření množiny.

Mapy

Mapy

Mapy (*asociativní pole*, nepřesně také hašovací tabulky nebo haše) fungují v podstatě na stejných principech a požadavcích jako *Set*:

- Ukládají ovšem dvojice (klíč, hodnota) a umožnují rychlé vyhledání dvojice podle hodnoty klíče.
- Základními metodami jsou: dotazy na přítomnost klíče v mapě (*containsKey*),
- výběr hodnoty odpovídající zadanému klíči (*get*),
- možnost získat zvlášť *množiny klíčů*, *hodnot* nebo *dvojic* (klíč, hodnota).

Mapy mají:

- podobné implementace jako množiny (tj. hašovací tabulky nebo stromy).
- logaritmickou složitost základních operací (*put*, *remove*, *containsKey*)

Uspořádané mapy

Uspořádané mapy:

- Implementují rozhraní *SortedMap* - API doc k rozhraní *SortedMap* [http://java.sun.com/j2se/1.5/docs/api/java/util/SortedMap.html]
- Dvojice (klíč, hodnota) jsou v nich *uspořádané podle hodnot klíče*.
- Existuje vestavěná impl. *TreeMap* - černobílé stromy (Red-Black Trees) - API doc ke třídě *TreeMap* [http://java.sun.com/j2se/1.5/docs/api/java/util/TreeMap.html]
- Uspořádání lze ovlivnit naprosto stejným postupem jako u uspořádané množiny.

Starší typy kontejnerů

Historie

Existují tyto starší typy kontejnerů (-> náhrada):

- *Hashtable -> HashMap*, *HashSet* (podle účelu)
- *Vector -> List*
- *Stack -> List*

Roli iterátoru plnil dříve *výčet* (*enumeration*) se dvěma metodami:

- *boolean hasMoreElements()*
- *Object nextElement()*

Srovnání implementací a odkazy

Srovnání implementací kontejnerů

Seznamy:

- na bázi pole (*ArrayList*) - rychlý přímý přístup (přes index)
- na bázi lineárního zřetězeného seznamu (*LinkedList*) - rychlý sekvenční přístup (přes iterátor)

téměř vždy se používá *ArrayList* - stejně rychlý a paměťově efektivnější

Množiny a mapy:

- na bázi hašovacích tabulek (*HashMap*, *HashSet*) - rychlejší, ale neuspořádané (lze získat iterátor procházející klíče uspořádaně)
- na bázi vyhledávacích stromů (*TreeMap*, *TreeSet*) - pomalejší, ale uspořádané
- spojení výhod obou - *LinkedHashSet*, *LinkedHashMap*

Odkazy

Demo efektivity práce kontejnerů - Demo kolekcí [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/kolekce/Kolekce.java] (beztypových/negenerických)

Velmi podrobné a kvalitní seznámení s kontejnery najdete na Trail: Collections [http://java.sun.com/docs/books/tutorial/collections/index.html]