Dynamické datové struktury (kontejnery)

Obsah

Kontejnery, iterátory, kolekce

- Kontejnery jako základní dynamické struktury v Jav#
- Kolekce, iterátory (Collection, Iterator)
- Seznamy (rozhraní List, t#ídy ArrayList, LinkedList)
- Množiny (rozhraní Set, t#ída HashSet), uspo#ádané množiny (rozhraní SortedSet, t#ída TreeSet), rozhraní Comparable, Comparator
- Mapy (rozhraní Map, t#ída HashMap), uspo#ádané mapy (rozhraní SortedMap, t#ída TreeMap)
- Klasické netypové vs. nové typové kontejnery - generické datové typy
- Iterace cyklem foreach
- Starší typy kontejner# (Vector, Stack, Hashtable)

Kontejnery

Co jsou kontejnery?

- dynamické datové struktury vhodné k ukládání prom#nného po#tu objekt# (p#esn#ji odkaz# na objekty)
- podle povahy kontejneru mohou mít r#zné specifické vlastnosti:

  seznamy — každý prvek v nich uložený má svou pozici (podobn# jako u pole, ale prvk# lze uložit potenciáln# neomezený po#et)

  množiny — prvek lze do množiny vložit nejvýš jedenkrát (odpovídá matematické p#edstav#), p#i porovnávání rozhoduje rovnost podle equals

  mapy — v kontejneru jsou dvojice (klí#, hodnota), p#i znalosti klí#e lze v kontejneru rychle najít hodnotu (p#ibližn# odpovídá databázovému klí#i)

- kontejnery jsou datové struktury v opera#ní pam#ti, nejsou automaticky "zálohované" do trvalé pam#ti (na disk)

Kontejnery

Kontejnery (containers) v Jav#

- slouží k ukládání objekt# (ne hodnot primitivních typ#!)
- v Jav# byly koncipovány jako *beztypové* - to už ale od verze Java 1.5 (tedy Java 5) neplatí!
- od Javy 5 mají kolekce tzv. *typové parametry* vyzna#ené ve špi#atých závorkách (nap#. List<Person>), jimiž ur#ujeme, jaké položky se do kolekce sm#jí dostat

V#tšinou se používají kontejnery hotové, vestav#né, tj. ty, jež jsou sou#astí Java Core API:

- vestav#né kontejnerové t#ídy jsou definovány v balíku `java.util` [http://www.google.com/search?q=java.util] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=java.util]
- je možné vytvo#it si vlastní implementace, obvykle ale zachovávající/implementující „standardní“ rozhraní

K #emu slouží?

- jsou dynamickými alternativami k poli a mají daleko širší použití
- k uchování prom#nného po#tu objekt# -
- po#et prvk# se v pr#b#hu existence kontejneru m#že m#nit
- oproti polím nabízejí #asov# efektivn#jší algoritmy p#ístupu k prvk#m

Základní kategorie kontejner#

Kategorie jsou dány tím, které rozhraní p#íslušný kontejner implementuje (základní jsou List<E> [http://java.sun.com/javase/6/docs/api/java/util/List.html], Set<E> [http://java.sun.com/javase/6/docs/api/java/util/Set.html] a Map<K, V> [http://java.sun.com/javase/6/docs/api/java/util/Map.html]).

Seznam (*List*)	lineární struktura, každý prvek má sv#j #íselný index (pozici)
Množina (*Set*)	struktura bez duplicitních hodnot a obecn# také bez uspo#ádání, umož#uje rychlé dotazování na p#ítomnost prvku
Asociativní pole, mapa (*Map*)	struktura uchovávající dvojice (klí#->hodnota), rychlý p#ístup p#es klí#

Kolekce

- jsou kontejnery implementující rozhraní *Collection* - API doc k rozhr. *Collection* [http://java.sun.com/javase/6/docs/api/java/util/Collection.html]
- Rozhraní kolekce popisuje velmi obecný kontejner, disponující operacemi: *p#idávání, rušení prvku, získání iterátoru, zjiš#ování prázdnosti* atd.
- Mezi kolekce pat#í mimo *Map*y všechny ostatní vestav#né kontejnery - *List*, *Set*
- Prvky kolekce nemusí mít svou pozici danou indexem - viz nap#. *Set*

Kontejnery - rozhraní, nepovinné metody

- Funkcionalita vestav#ných kontejner# je obvykle p#edepsána výhradn# *rozhraním*, jenž implementují.
- Rozhraní však p#ipoušt#jí, že n#které metody jsou *nepovinné*, t#ídy jej nemusí implementovat!
- V praxi se totiž n#kdy nehodí implementovat jak #tecí, tak i zápisové operace, protože n#které kontejnery chceme mít „read-only“

Iterátory

Iterátory jsou prost#edkem, jak sekven#n# procházet prvky kolekce bu#to

- v neur#eném po#adí nebo
- v uspo#ádání (u uspo#ádaných kolekcí)

Každý iterátor musí implementovat velmi jednoduché rozhraní Iterator<E> [http://java.sun.com/javase/6/docs/api/java/util/Iterator.html] se t#emi metodami:

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

- *void remove()*

Seznamy

Seznamy

- lineární struktury
- implementují rozhraní *List* (rozší#ení *Collection*) API doc k rozhr. *List* [http://java.sun.com/javase/6/docs/api/java/util/List.html]
- prvky lze adresovat indexem (typu int)
- poskytují možnost získat dop#edný i zp#tný *iterátor*
- lze pracovat i s *podseznamy*

Množiny

Množiny

Množiny

- jsou struktury standardn# bez uspo#ádání prvk# (ale existují i uspo#ádané, viz dále)
- implementují rozhraní *Set* (což je rozší#ení *Collection*)

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

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

Standardní implementace množiny:

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

Množiny - implementace

Standardní implementace množiny:

hašovací tabulky	HashSet [http://java.sun.com/javase/6/docs/api/java/util/HashSet.html] - potenciáln# rychlejší, ale neuspo#ádané hodnoty
#ernobílé stromy	TreeSet [http://java.sun.com/javase/6/docs/api/java/util/TreeSet.html] - uspo#ádané hodnoty, s garantovanou logaritmickou složitostí

Uspo#ádané množiny

Uspo#ádané množiny:

- Implementují rozhraní *SortedSet* - API doc k rozhraní *SortedSet* [http://java.sun.com/javase/6/docs/api/java/util/SortedSet.html]
- Jednotlivé prvky lze tedy iterátorem procházet v p#esn# definovaném po#adí - uspo#ádání podle *hodnot prvk#*.
- Existuje vestav#ná implementace *TreeSet* - #ernobílé stromy (Red-Black Trees) API doc ke t#íd# *TreeSet* [http://java.sun.com/javase/6/docs/api/java/util/TreeSet.html]

Uspo#ádání je dáno bu#to:

- standardním chováním metody *compareTo* vkládaných objekt# - pokud implementují rozhraní *Comparable*
- nebo je možné uspo#ádání definovat pomocí tzv. *komparátoru* (objektu impl. rozhraní *Comparator*) poskytnutých p#i vytvo#ení množiny.

Mapy

Mapy

Mapy (*asociativní pole*, nep#esn# také hašovací tabulky nebo haše) fungují v podstat# na stejných principech a požadavcích jako *Set*:

- Ukládají ovšem dvojice (klí#, hodnota) a umožnují rychlé vyhledání dvojice podle hodnoty klí#e.
- Základními metodami jsou: dotazy na p#ítomnost klí#e v map# (*containsKey*),
- výb#r hodnoty odpovídající zadanému klí#i (*get*),
- možnost získat zvlášt *množiny klí##*, *hodnot* nebo *dvojic* (klí#, hodnota).

Mapy - implementace a složitosti

Mapy mají podobné implementace jako množiny, tj.:

hašovací tabulky	HashMap [http://java.sun.com/javase/6/docs/api/java/util/HashMap.html] - potenciáln# rychlejší, ale neuspo#ádané klí#e

#ernobílé stromy	TreeMap [http://java.sun.com/javase/6/docs/api/java/util/TreeMap.html] - uspo#ádané klí#e, s garantovanou logaritmickou složitostí

Složitost základních operací (*put*, *remove*, *containsKey*):

- Mapy implementované jako stromy mají nejvýš logaritmickou složitost základních operací.
- U map implementovaných hašovací tabulkou složitost v praxi závisí na kvalit# hašovací funkce (metody hashCode) na ukládáných objektech, teoreticky se blíží složitosti konstantní.

Uspo#ádané mapy

Uspo#ádané mapy:

- Implementují rozhraní *SortedMap* - API doc k rozhraní *SortedMap* [http://java.sun.com/javase/6/docs/api/java/util/SortedMap.html]
- Dvojice (klí#, hodnota) jsou v nich *uspo#ádané podle hodnot klí#e*.
- Existuje vestav#ná implementace *TreeMap* - #ernobílé stromy (Red-Black Trees) - API doc ke t#íd# *TreeMap* [http://java.sun.com/javase/6/docs/api/java/util/TreeMap.html]
- Uspo#ádání lze ovlivnit naprosto stejným postupem jako u uspo#ádané množiny.

Další informace a odkazy

Kontejnery - soub#žný p#ístup, výjimky

- Moderní kontejnery jsou *nesynchronizované*, nep#ipoušt#jí soub#žný p#ístup z více vláken.
- Standardní, nesynchronizovaný, kontejner lze však „zabalit“ synchronizovanou obálkou.
- P#i práci s kontejnery m#že vzniknout #ada *výjimek*, nap#. *IllegalStateException* apod.
- V#tšina má charakter výjimek *b#hových*, není povinností je odchytávat - pokud v##íme, že nevzniknou.

Starší typy kontejner#, Enumeration

Existují tyto starší typy kontejner# (za -> uvádíme náhradu):

- *Hashtable -> HashMap, HashSet* (podle ú#elu)
- *Vector -> List*
- *Stack -> List*

Roli iterátoru plnil d#íve *vý#et* (Enumeration

http://java.sun.com/javase/6/docs/api/java/util/Enumeration.html]) se dv#ma metodami:

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

Srovnání implementací kontejner#

Seznamy:

- na bázi pole (*ArrayList*) - rychlý p#ímý p#ístup (p#es index)
- na bázi lineárního z#et#zeného seznamu (*LinkedList*) - rychlý sekven#ní p#ístup (p#es iterátor)

tém## vždy se používá *ArrayList* - stejn# rychlý a pam##ov# efektivn#jší

Množiny a mapy:

- na bázi hašovacích tabulek (*HashMap*, *HashSet*) - rychlejší, ale neuspo#ádané (lze získat iterátor procházející klí#e uspo#ádan#)
- na bázi vyhledávacích strom# (*TreeMap*, *TreeSet*) - pomalejší, ale uspo#ádané
- spojení výhod obou - *LinkedHashSet*, *LinkedHashMap*

Odkazy

Demo efektivity práce kontejner# - Demo kolekcí [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/kolekce/Kolekce.java] (beztypových, nepoužívajících generické typy)

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