ArrayList vs. LinkedList

Java Syntax (/quests/QUEST_JAVA_SYNTAX)
8 уровень (/quests/lectures/?quest=QUEST_JAVA_SYNTAX&level=8), 5 лекция (/quests/lectures/questsyntax.level08.lecture05)

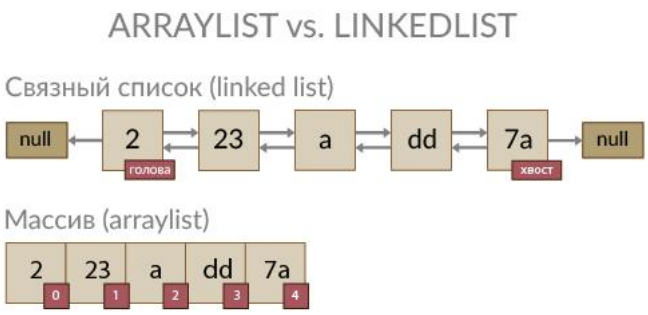
ОТКРЫТА

— Как насчёт немного размять мозги? Надеюсь, они ещё не закипели.

— В таблице контейнеров и коллекций ты ранее видел, что у одного и того же интерфейса может быть несколько реализаций. Сейчас я расскажу тебе, зачем это нужно. И в чем отличие ArrayList от LinkedList.

— Все дело в том, что коллекции могут быть реализованы разными способами и нет единственного – самого правильного. При одном подходе одни операции являются быстрыми, а остальные медленными, при другом – все наоборот. Нет одного идеального, подходящего всем решения.

— Поэтому было решено сделать несколько реализаций одной и той же коллекции. И каждая реализация была оптимизирована для какого-то узкого набора операций. Так появились разные коллекции. Давай рассмотрим это на примере двух классов – ArrayList и LinkedList.



— ArrayList реализован внутри в виде обычного массива. Поэтому при вставке элемента в середину, приходиться сначала сдвигать на один все элементы после него, а уже затем в освободившееся место вставлять новый элемент. Зато в нем быстро реализованы взятие и изменение элемента – операции get, set, так как в них мы просто обращаемся к соответствующему элементу массива.

— LinkedList реализован внутри по-другому. Он реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Чтобы вставить элемент в середину такого списка, достаточно поменять ссылки его будущих соседей. А вот чтобы получить элемент с номером 130, нужно пройтись последовательно по всем объектам от 0 до 130. Другими словами операции set и get тут реализованы очень медленно. Посмотри на таблицу:

Описание	Операция	ArrayList	LinkedList
Взятие элемента	get	Быстро	Медленно

Присваивание элемента

set

Быстро

Медленно

Добавление элемента

add

Быстро

Быстро

Вставка элемента

add(i, value)

Медленно

Быстро

Удаление элемента

remove

Медленно

Быстро

— Ага. Кое-что начинает проясняться. А есть какие-нибудь критерии или правила, когда какая коллекция лучше?

— Ну, для простоты, я бы сформулировала такое правило: если ты собираешься вставлять (или удалять) в середину коллекции много элементов, то тебе лучше использовать LinkedList. Во всех остальных случаях – ArrayList.

— Как они устроены мы разберем в старших уровнях, а пока будем учиться ими пользоваться.

‹ (/quests/lectures/questsyntax.level08.lecture04)

×8 › (/quests/lectures/questsyntax.level08.lecture06)

0 0 0 0 0 +33

Комментарии (27) популярные новые старые

Никита

Введите текст комментария

Den 14 уровень, Одесса 23 февраля, 14:25

че-то подвис...

"А вот чтобы получить элемент с номером 130, нужно пройтись последовательно по всем объектам от 0 до 130"

а чтобы вставить или удалить элемент с номером 130, LinkedList не должен пройтись по всем объектам до 130го? как он БЫСТРО находит 130й элемент?

или БЫСТРО в таблице имеется в виду быстрее чем ArrayList, который должен после удаления/вставки двигать все элементы после 130го?

Ответить 0

Радхараман Скороход 13 уровень 15 марта, 17:25

Имеется ввиду, что для ArrayList пройтись до 130 элемента можно очень быстро, но переписывать "ручками" еще столько же элементов после него - согласитесь, это гораздо дольше. А если их там не 130, а 1300? 13000? В любом случае все придется переписывать. Для LinkedList найти 130 элемент дольше, чем для ArrayList, но зато скорость операций "найти" и "изменить"практически одинаковая. Кстати это и памяти требует меньше примерно в два раза.

Ответить 0

Олег 14 уровень 16 февраля, 23:54

Почитал хабри и пришел к выводу, что LinkedList уступает ArrayList и ним толком никто и не пользуется, ибо он уступает по скорости во всем. Непонятно тогда к чему здесь пишут, что Linked в чем-то быстрее. Даже создатель LinkedList в твиттере написал

Ответить +10

Fs Jt 20 уровень, Киев 28 ноября 2017, 20:22

ну не знаю. измерил на 100 тысячах взятий/присваиваний и т.д., вот что вышло (результат в мс):

ArrayList Add: 6
ArrayList Set: 4
ArrayList Get: 3
ArrayList Remove: 2367

LinkedList Add: 6
LinkedList Set: 3
LinkedList Get: 10304
LinkedList Remove: 3

Ответить +4

Кутх 11 уровень 2 декабря 2017, 22:24

А у меня такие на 100 тысячах:

Time of ArrayList: add - 252 ms; get - 2 ms; set - 4 ms; remove - 220 ms
Time of LinkedList: add - 6118 ms; get - 6008 ms; set - 5888 ms; remove - 6685 ms

p.s. add и remove делал в середине списка, а get и set проходился по всему списку.

Ответить +2

Кутх 11 уровень 2 декабря 2017, 22:45

Если же add делать в конец списка (как более типовая операция), а remove первого элемента списка, то результаты такие:

Time of ArrayList: add - 7 ms; get - 5 ms; set - 9 ms; remove - 517 ms
Time of LinkedList: add - 5 ms; get - 4595 ms; set - 4703 ms; remove - 3 ms

У метода add разница в пользу LinkedList, но совсем незначительная.
А в методе remove LinkedList выигрывает только при удалении первого элемента списка (при удалении последнего результаты идентичны - 3 мс. Отсюда вывод: "ArrayList быстрее чистить с конца").

Т.о., по моим замерам, можно сказать, ArrayList выигрывает во всех методах, т.к. суммарно работать быстрее программа с LinkedList будет, если только добавить последовательно элементы в список, а потом их не считывая из списка, удалить.

Ответить 0

Дмитрий Пивоваров 22 уровень 6 февраля, 11:35

Погоняйте с add в серединку, типа add(50000, "bla bla bla") вот там интересно

Ответить 0

Радхараман Скороход 13 уровень 16 марта, 14:33

Ну, например на миллионе add в начало:
Время для ArrayList - 146268
Время для LinkedList - 599

Ответить 0

Богдан 10 уровень, Киев 28 ноября 2017, 19:45

Так вот оно что!
List/Set/Map - это интерфейсы, а ArrayList, LinkedList,HashSet,HashMap,TreeSet,TreeMap и пр. - это конкретные реализации этих интерфейсов.
Т.е. полиморфизм в чистом, мать его, виде:)
Дошло.
Спасибо

Ответить +30

Влад Бутенко 21 уровень 9 ноября 2017, 22:46

Большинство программистов говорят, что за всю свою карьеру почти не использовали ЛинкедЛист, т.к. он просто напросто МЕДЛЕННЕЕ ВО ВСЕМ. ЭрэйЛист обгоняет его даже в случае удаления и добавления элементов, поэтому, ребята, не парьтесь и юзайте всегда именно ArrayList:)

Ответить +6

Андрей Селезнев 33 уровень, Кубань 2 октября 2017, 09:43

Я таки не понимаю, с чего это добавление или удаление элементов в середину линкедлиста будет быстрым. Само удаление операция действительно быстрая. Надо только перенаправить ссылки соседних элементов. Но ведь сначала нужно найти удаляемый элемент по индексу. А для этого пробежаться по всему списку от головы или хвоста. А это уже чложность O(n). И с добавлением та же ситуация. Надо сначала найти место куда встаылять. И это тоже O(n).

Ответить +4

Artem Ostretsov 14 уровень 10 октября 2017, 07:28

Короче, профит в потребляемой памяти, а не то, что тут написано. Представь пустой массив[1000] с двумя элементами в середине. Для редких данных (матрицы там какие-нибудь с нулями) лучше LinkedList.

Ответить +3

Maxim 16 уровень 30 сентября 2017, 23:49

скажите в чем разница между set и add?

Ответить 0

Андрей Кожарин 16 уровень 1 октября 2017, 14:33

Set - изменяет значение определённого элемента в списке,но ничего не добавляет

Ответить +1

NEMEHANIK 10 уровень 9 июля 2017, 19:55

Методы set и add(i, value) у списков типа LinkedList выполняются, на мой взгляд, приблизительно одно и тоже время (если я правильно понял логику его работы). В таблице правильнее было бы расставить слова БЫСТРЕЕ и МЕДЛЕННЕЕ, подчеркивая отношение к аналогичным методам другого типа списка.

Ответить 0

Роман 12 уровень 17 августа 2017, 16:11

Если быть точным, то при добавлении 1 элемента списки работают одинаково, но если у нас массив в 2 млн элементов и нужно добавить еще 200 тыс, то LinkedList будет работать быстрее. ArrayList при добавлении большого количества элементов запустит создание нового массива на бОльшее количество элементов, копирование из старого в новый и удаление старого и только после всего этого добавит новый список.

Ответить +3

Евгений 15 уровень 27 августа 2017, 03:06

Речь идет не о добавлении (что значит в конец списка), а о вставке (в середину списка).
Даже при вставке 1 элемента, ArrayList будет дольше возиться, чем LinkedList.
Частный случай, если размер списков совсем меленький - первого порядка.

Ответить 0

Sergio 19 уровень 29 августа 2017, 10:40

Эти 200 тыс. элементов можно еще по-разному вставить. Если у нас цикл с 2000 итераций, в котором мы всякий раз вставляем в ArrayList по 100 элементов, это будет очень медленно, поскольку придется каждый раз массив заново пересоздавать. LinkedList такую процедуру переживет намного проще.

Ответить 0

Кутх 11 уровень 2 декабря 2017, 22:57

Мои замеры показывают, что на 100000 итераций вставки в середину списка ArrayList вставляет почти в 30 раз быстрее.

```
1  for (int i = 0; i < 100000; i++) {
2      list.add((int) i/2, new Object());
3  }
```

Ответить +1

Андрей Кузнецов 20 уровень, Москва 19 июня 2017, 22:46

Может быть хвост есть всё, что не голова?

Ответить 0

Michael 27 уровень, Санкт-Петербург 9 сентября 2017, 14:09

В двусвязном списке крайние элементы являются головой и хвостом, ибо они указывают, что дальше идти некуда. А вот почему они изобразили хвост не в самом конце - это я не знаю. Может в джаве есть какие-то уникальные особенности :\

Ответить +2

Aleksandr S Karasev 10 уровень 14 июня 2017, 12:26

Это же схематично нарисовано. Зрите в корень! Смысл в том, чтоб вы понимали, что отсчет всегда будет с головы.
Поправьте, если я не прав.

Ответить 0

Загрузить еще