*Курочкина Е.С.*

1. **Сравнение производительности MyLinkedList и java.util.LinkedList.**

/Единицы измерения: наносекунды/

|  |  |  |
| --- | --- | --- |
|  | MyLinkedList | LinkedList |
| Add | 3500 | 2700 |
| Add by index | 3300 | 8100 |
| Get by index | 3700 | 7200 |
| Remove | 2100 | 7600 |
| Clear | 95900 | 2600 |

Написанный мной MyLinkedList имеет гораздо большую производительность в части выполнения операции clear. В операциях add by index, get, remove он также стал более эффективным, уступив Java реализации лишь только в выполнении add.

1. **Сравнительный анализ производительности основных Java коллекций.**

/Единицы измерения: наносекунды/

1. **Сравнить производительность ArrayList и LinkedList для основных операций**
2. *Операции добавления и очистки:*

Операция add наиболее быстро выполнена в LinkedList, а clear – в ArrayList.

1. *Операции вставки по индексу, получения, удаления элемента:*

В данном случае самыми долгими операции add, get, remove проявили себя в ArrayList.

|  |  |  |
| --- | --- | --- |
|  | ArrayList | LinkedList |
| Add | 5250000 | 4122100 |
| Add by index | 54400 | 18000 |
| Get by index | 4800 | 5500 |
| Remove by index/key | 46000 | 3500 |
| Remove by object | 38300 | 8000 |
| Clear | 944500 | 2335900 |

**ArrayList** внутри реализован как обычный массив.

+ элементы расположены друг за другом;

+ быстрые операции get и set, т.к. доступ осуществляется по индексу;

- При вставке элемента в середину приходится сдвигать на один все элементы после него, затем в свободное место вставлять элемент. Это дольше, чем в LinkedList.

**LinkedList** внутри реализован как связный список. Элементы друг с другом связаны, но находятся в разных ячейках памяти;

+ удобно вставлять элементы в середину - меняются лишь ссылки на соседей;

- неудобно использовать get и set, т.к. нужно проходиться по всем элементам. Это долго.

Почти во всех случаях советуется использовать ArrayList. LinkedList будет полезен, если часто нужно вставлять элементы не в начало или конец.

1. **Сравнить производительность HashSet, LinkedHashSet, TreeSet для основных операций**
2. *Операции добавления и очистки:*

Операция add в TreeSet оказалась самой долгой, однако операция clear в том же TreeSet стала наиболее быстрой.

1. *Операции вставки по индексу, получения, удаления элемента:*

Как можно заметить, TreeSet снова оказался обладателем самых долгих операций contains и remove. Производительность в данном случае по всем параметрам выше у LinkedHashSet.

|  |  |  |  |
| --- | --- | --- | --- |
|  | HashSet | LinkedHashSet | TreeSet |
| Add | 10232600 | 7170500 | 37258500 |
| Contains | 16400 | 2600 | 69700 |
| Remove by object | 700 | 500 | 2300 |
| Clear | 2053900 | 317700 | 6600 |

Таким образом, на первом месте по производительности оказался LinkedHashSet. Следом идет HashSet.

Если нам нужен гарантированный порядок элементов – используем **LinkedHashSet**.

Если нужно, чтобы каждый раз добавлялся в новый bucket – берем **HashSet**.

**TreeSet** лучше использовать, если нужна упорядоченная структура данных и важен быстрый поиск.

1. **Сравнить производительность HashMap, LinkedHashMap, TreeMap**
2. *Операции добавления и очистки:*

Наибольшую производительность в выполнении операции put показал LinkedHashMap. Clear – TreeMap.

1. *Операции вставки по индексу, получения, удаления элемента:*

TreeMap потратил больше всего наносекунд на выполнение remove. LinkedHashMap лидировал по производительности в операциях get и remove.

|  |  |  |  |
| --- | --- | --- | --- |
|  | HashMap | LinkedHashMap | TreeMap |
| Put | 7756000 | 877830 | 26373800 |
| Get by key | 9900 | 3700 | 7400 |
| Remove by key | 3000 | 2500 | 2300 |
| Remove by value | 7000 | 6100 | 32200 |
| Clear | 99500 | 196900 | 300 |

**Используем:**

**HashMap** – если все равно на порядок элементов при обходе; если нас устраивает, что при повторном добавлении элемента с тем же ключом первое значение будет утеряно.

**LinkedHashMap** – если важно сохранить порядок добавления элементов;

**TreeMap** – если нужен отсортированный по ключам набор элементов.



 