**Сравнительный анализ производительности основных Java коллекций.**

1. **Сравнить производительность ArrayList и LinkedList для основных операций**

**Тесты проводятся на 100\_000 элементов**

|  |  |  |
| --- | --- | --- |
|  | **ArrayList** | **LinkedList** |
| **Добавление в начало** | -1482.5865ms | **-18.152601ms** |
| **Добавление в середину** | **3282.8537ms** | 9554.7512ms |
| **Добавление в конец** | **11.227499ms** | 12.978701ms |
| **Удаление в начале** | 1555.698801ms | **8.5061ms** |
| **Удаление в середине** | **3576.5408ms** | 3684.605ms |
| **Удаление в конце** | -3.242699ms | **-1.9177ms** |
| **Получение в начале** | **0,0005ms** | 0.0191ms |
| **Получение в середине** | **0,0006ms** | 0.2803ms |
| **Получение в конце** | **0,0008ms** | -1.1211ms |

**ArrayList** следует использовать , когда в основном нам необходим доступ по индексу и мы не так часто добавляем новые элементы. Также добавление в середину на этом примере происходит быстрее , поскольку LinkedList тратит много времени на поиск элемента с необходимым индексом , пусть и ArrayList приходится сдвигать элементы.

**LinkedList** стоит же применять , когда мы часто вставляем и удаляем элементы.

1. **Сравнить производительность HashSet, LinkedHashSet, TreeSet для основных операций**

**Тесты проводятся на 1000\_000 элементов**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **HashSet** | **TreeSet** | **LinkedHashSet** |
| **Добавление** | **370.7952** | 604.4006ms | 408.9897ms |
| **Удаление** | 159.6384ms | 185.2717ms | **74.507ms** |
| **Получение** | 0.0102ms | 0.0568ms | **0.0019ms** |

Использование **HashSet** будет уместно , когда нам необходимо добавлять большие объёмы данных и хранить их.

Использование **TreeSet** будет уместно , когда нам постоянно необходимо работать   
Использование же **LinkedHashSet** уместно, когда нам необходимо часто брать элементы по индексу.

1. **Сравнить производительность HashMap, LinkedHashMap, TreeMap**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **HashMap** | **TreeMap** | **LinkedHashMap** |
| **Добавление** | **331.0533ms** | 397.377201ms | 337.1853ms |
| **Удаление** | **49.8824ms** | 148.246999ms | 74.507ms |
| **Получение** | 0.0102ms | 0.0568ms | **0.0019ms** |

Использование HashMap удобно в случаях , когда нам не важен порядок хранения элементов и мы хотим максимально быстро добавлять и удалять элементы.

Использование TreeMap удобно в случаях , когда нам необходимо хранить в определенном порядке элементы и самостоятельно выставлять этот порядок.

Использование **LinkedHashMap** удобно тем, что мы можем сохранить порядок ввода данных и при том осуществлять быстрое взятие по ключу.