Потокобезопасность контейнеров в Java

Можно выделить такие основные методы реализации потокобезопасности для **контейнеров** Java:

* Использование внешней синхронизации;
* Использование специальных потокобезопасных коллекций;

Синхронизация и потокобезопасность это разные вещи. Синхронизация это одна из целей потокобезопасности в целом. Здесь эти понятия не акцентируются и не уточняются, поскольку они часть другой большой темы.

Нужно исследовать как Java поддерживает стандартные средства управления потоками и процессами:

* Критические секции;
* Мютексы;
* Семафоры;
* События;

**Внешняя синхронизация**

Java предоставляет ключевое слово **synchronized** которое предотвращает параллельный доступ к каким либо ресурсам из разных потоков. Блок кода, метод или переменная могут быть объявлены как **synchronized**. Это значит, что если два потока одновременно пытаются выполнить **synchronized** метод, то второй поток будет ждать пока первый не завершит выполнение метода.

Синтаксис использования **synchronized**:

* Для блока кода

synchronized(objectidentifier) {

// Access shared variables and other shared resources

}

* Для метода

public synchronized void test(String name) {  
 …

}

Для использования **synchronized** с коллекциями из пакета java.util нужно сначала получить потокобезопасный вариант объекта с помощью одного из статических методов класса Collections. эти методы блокируют **доступ ко всей коллекции**.

Далее реализуем ручную синхронизацию с помощью **synchronized**:

public static void main(String args[]) {

HashMap<Integer, String> hmap= new HashMap<Integer, String>();

hmap.put(2, "Anil");

hmap.put(44, "Ajit");

hmap.put(1, "Brad");

hmap.put(4, "Sachin");

hmap.put(88, "XYZ");

Map map= Collections.synchronizedMap(hmap);

Set set = map.entrySet();

synchronized(map){

Iterator i = set.iterator();

// Display elements

while(i.hasNext()) {

Map.Entry me = (Map.Entry)i.next();

System.out.print(me.getKey() + ": ");

System.out.println(me.getValue());

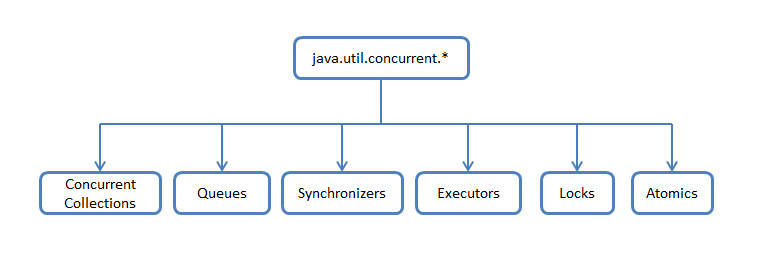
}

}

}

Класс Collections содержит методы-обертки для получения всех типов потокобезопасных коллекций.

**Потокобезопасные коллекции**

Потокобезопасные коллекции реализованы в пакете java.util.concurrent.\*. Схематично можно поделить классы и интерфейсы по функциональному признаку таким образом:  
  


**Concurrent Collections** - набор коллекций, более эффективно работающие в многопоточной среде нежели стандартные универсальные коллекции из java.util пакета. Например вместо базовой обертки Collections.synchronizedList() с блокированием доступа ко всей коллекции используются блокировки по сегментам данных или же оптимизируется работа для параллельного чтения данных по [wait-free](http://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B0%D1%8F_%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) алгоритмам.  
  
**Queues** - неблокирующие и блокирующие очереди с поддержкой многопоточности. Неблокирующие очереди заточены на скорость и работу без блокирования потоков. Блокирующие очереди используются, когда нужно «притормозить» потоки «Producer» или «Consumer», если не выполнены какие-либо условия, например, очередь пуста или перепонена, или же нет свободного «Consumer»'a.  
  
**Synchronizers** - вспомогательные утилиты для синхронизации потоков. Представляют собой мощное оружие в «параллельных» вычислениях.  
  
**Executors** - содержит в себе отличные фрейморки для создания пулов потоков, планирования работы асинхронных задач с получением результатов.   
  
**Locks** - представляет собой альтернативные и более гибкие механизмы синхронизации потоков по сравнению с базовыми synchronized, wait, notify, notifyAll.  
  
**Atomics** - классы с поддержкой атомарных операций над примитивами и ссылками.

Hastable и java.util.concurrent

Нужно сразу заметить, что Hastable считается устаревшим и не является частью современного [Java Collections Framework](http://docs.oracle.com/javase/7/docs/technotes/guides/collections/index.html). В отличие от новых реализаций коллекций, Hashtable синхронизирован. Но если многопочточность не нужно, то лучше использовать [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) вместо Hashtable. Если многопоточность нужна, то лучше использовать [ConcurrentHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html) вместо Hashtable [2].

**Разница между HashMap и Hashtable**

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **HashMap** | **Hashtable** | **Коментарий** |
| Synchronized | No | Yes |  |
| Thread-Safe | No | Yes |  |
| Null Keys and Null values | One null key ,Any null values | null keys and values not permitted |  |
| Iterator type | fail-fast iterator | fail-safe iterator | Hashmap object values are iterated by using iterator .HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.  Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification  of the collection  Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure. Any structural modification done to the iterator affects the copied data structure. |
| Performance | Fast | Slow in comparison with HashMap |  |
| Superclass | AbstractMap | Dictionary |  |
| Legacy | No | Yes | Better to use [ConcurrentHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html) in place of Hashtable |

1. <http://habrahabr.ru/company/luxoft/blog/157273/>
2. <http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html>
3. <http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html>