Google olabi

<HASHTABLES>

{ Introdução

Quase toda linguagem de programação contém estruturas de dados baseadas em *hash*. A linguagem java contém estruturas de dados de tabela *hash* (Hashtable), mapa de *hash* (HashMap) e árvores baseadas na função *hash*. A base destas estruturas de dados é o design do valor-chave: nele, cada chave é única, mas o mesmo valor pode existir para várias chaves correspondentes a um objeto, uma vez que esta função sempre retorna o valor inteiro para o mesmo objeto. Destrinchamos todos estes conceitos e terminologias a seguir.

{ O que é hash?

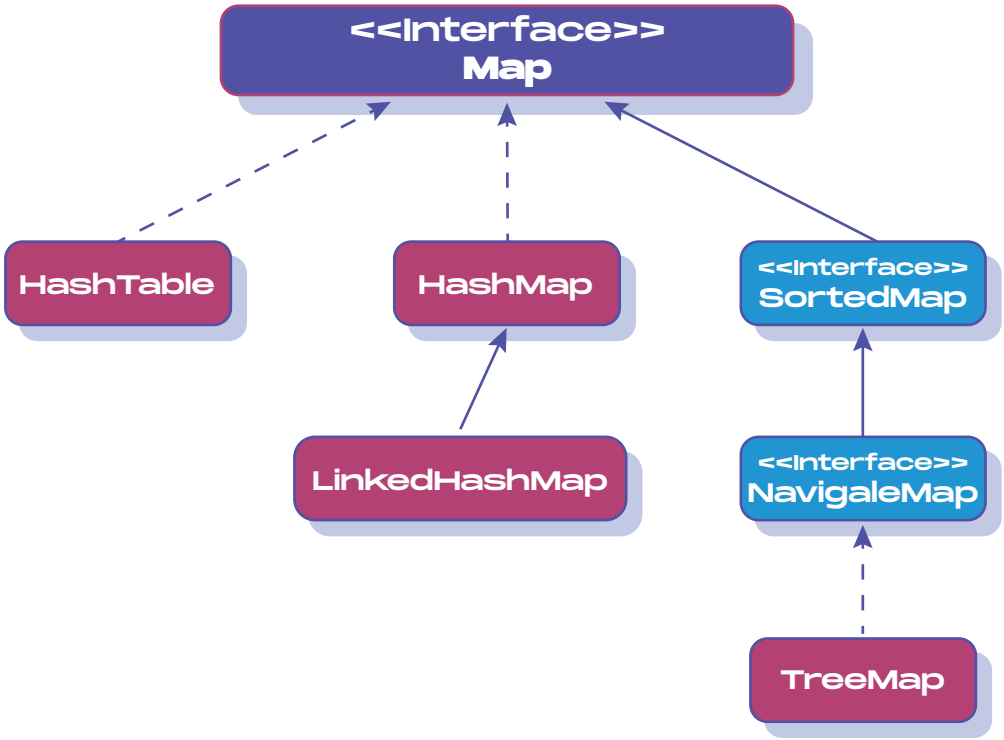
Hashing é o processo de transformar qualquer chave ou *string* de caracteres em outro valor, usualmente representado por um valor ou chave de comprimento fixo mais curto de forma a facilitar a localização ou uso da *string* original. Uma função *hash* gera novos valores baseados em um algoritmo matemático, também conhecido como "valor de *hash*". Para evitar que a conversão do *Hash* volte à chave original, o valor sempre utiliza algoritmo unidirecional.

O *hashing* também permite o acesso a dados de maneira eficiente no que diz respeito à computação e espaço de armazenamento, já que reduz o tempo de acesso não linear de listas ordenadas, não ordenadas e de árvores estruturadas, bem como requisitos de armazenamento - frequentemente exponenciais, de acesso direto à espaços de estado de chaves grandes, ou de comprimento variável.

{ Map e HashMap

Map é a interface usada para armazenar dados em um par chave-valor e HashMap, a classe de implementação da interface. Existem várias classes de Map em java que implementam a interface para armazenamento de dados em um par de valor-chave, como TreeHashMap e LinkedHashMap. Sozinha, a interface não pode ser usada para armazenamento de dados, mas, ainda assim, ela permite a criação de um objeto a partir de suas classes de implementação - que, por sua vez, utilizam a referência Map para armazenar o objeto.

{ Hierarquia da interface Map



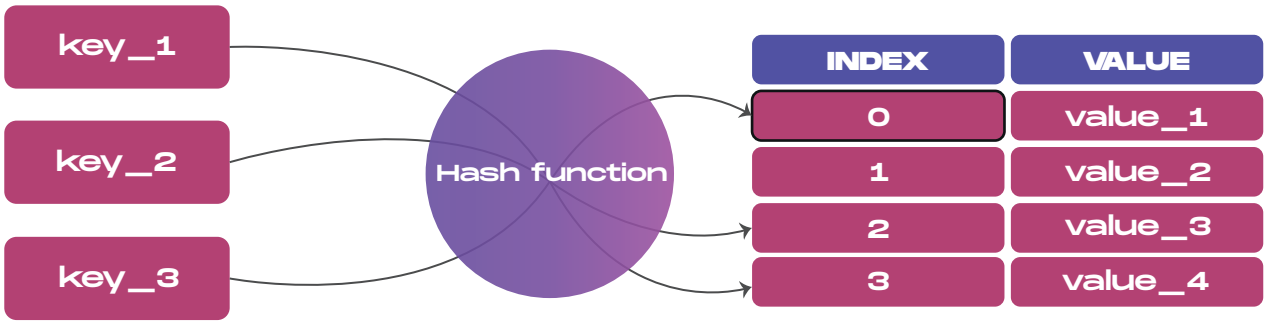
O uso mais popular de *hash* se dá na implementação de tabelas (Hashtables). As funções de *hash* e suas tabelas associadas são utilizadas em aplicativos de armazenamento e recuperação de dados de forma a permitir que eles sejam acessados em pouco tempo e de maneira quase constante. Estas funções exigem um espaço de armazenamento uma fração maior do que o total necessário para os próprios dados ou registros.

{ Hashtable

Uma tabela de hash armazena pares de chave e valor em uma lista - acessível por meio do índice. Como os pares de chave e valor são ilimitados, a função hash mapeia as chaves de acordo com o tamanho da tabela. Assim, um valor de hash torna-se o índice de um elemento específico.

A classe Hashtable implementa uma tabela que mapeia chaves para valores herdando a classe Dictionary e implementando a interface Map. Digamos que temos um dicionário no qual cada palavra possui sua definição. Como precisamos obter, inserir e remover palavras deste dicionário rapidamente, elas se tornam as chaves na Hashtable, uma vez que são únicas. Os valores ficam por conta das definições.

<Exemplo de Hashtable>



Para armazenar e recuperar com sucesso objetos de uma tabela *hash*, os objetos usados como chaves precisam implementar o método **hashCode()** para determinar qual contêiner o par chave-valor deverá mapear. Usualmente, o hashCode é um inteiro não-negativo comum a objetos iguais, que pode ou não ser igual em objetos desiguais. Para determinar se dois objetos são iguais ou não, o hashCode faz uso do método **equal().**

{ Colisão de hash

Quando dois objetos desiguais possuem o mesmo hashCode, ocorre uma colisão. Para resolvê-la, a tabela utiliza um *array* de listas no qual os pares mapeados para um único contêiner (*array index*) são armazenados em uma lista. A referência da lista é armazenada no *array index*.

{ Diferenças entre HashMap e Hashtable

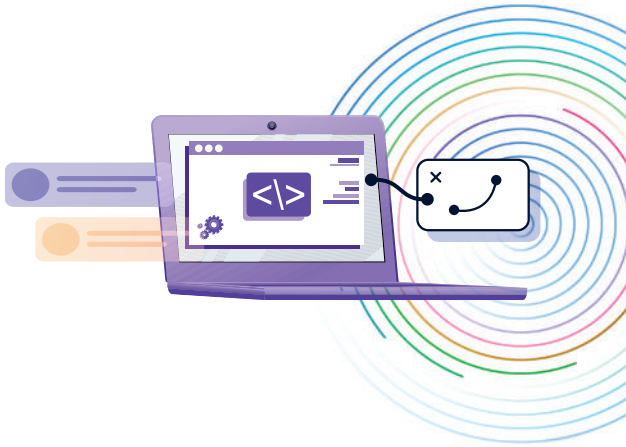
HashMap e Hashtable são usados para armazenar dados em forma de chave e valor por meio de *hashing*, que armazena chaves exclusivas. Entretanto, são muitas as diferenças entre estas duas classes:

HashMap	Hashtable
Não é sincronizado, nem seguro, e tão pouco pode ser compartilhado em muitas threads sem código de sincronização adequado	É sincronizada e segura, o que a torna compartilhável em muitas threads
Permite chave e múltiplos valores nulos	Não permite chave ou valor nulos
Nova classe introduzida em JDK 1.2	Classe clássica
É rápido	É lenta
É percorrido pelo Iterator	É percorrida pelo Enumerator e pelo Iterator
Iterator rápido em caso de falha	Enumerator não é rápido em caso de falha
Herda a classe AbstractMap	Herda a classe do Dicionário

{ Métodos de Hashtable

Alguns dos métodos utilizados para Hashtable incluem:

Método	Descrição
clear()	Limpa a Hashtable
clone()	Cria uma cópia
compute()	Calcula um mapeamento para a chave especificada e seu valor mapeado atual
containsKey()	Testa se o objeto especificado é uma chave
containsValue()	Retorna verdadeiro se esta Hashtable mapear uma ou mais chaves para este valor
elements()	Retorna uma enumeração dos valores
get()	Retorna o valor para o qual a chave especificada é mapeada
isEmpty()	Testa se a Hashtable não contém nenhuma chave para os valores
keys()	Retorna uma enumeração das chaves
merge()	Se a chave especificada não estiver associada a um valor, ou está associada a um valor nulo, ela se associa ao valor não-nulo
remove()	Remove a chave e seu valor correspondente
size()	Retorna o número de chaves



Referências

https://www.geeksforgeeks.org/hashtable-in-java/

https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html

https://stringfixer.com/pt/Hash_sum

https://www.javatpoint.com/difference-between-hashmap-and-hashtable

Hash

https://pt.education-wiki.com/7864460-hashing-function-in-java

https://www.techtarget.com/searchdatamanagement/definition/hashing

Map e HashMap

https://www.delftstack.com/pt/howto/java/difference-between-hashmap-and-map-in-java/