Universidade Federal do Ceará - Campus de Quixadá

Programação para Design

Coleções

Prof. Anderson Lemos

Introdução

- São estruturas de dados que tem como objetivo “colecionar” os dados de diversas formas.
- Em TypeScript, são oferecidas abstrações de alto nível para se trabalhar com essas estruturas. Essas abstrações facilitam:
  - Formas de organização dos dados
  - Formas de acesso, busca, inserção e remoção.
  - Eficiência.

Coleções

- As coleções compreendem:
  - Iteradores
  - Vetores e Listas
  - Conjuntos
  - Pilhas/Filas
  - Árvores binárias e tabelas Hash
  - Etc.

Coleções

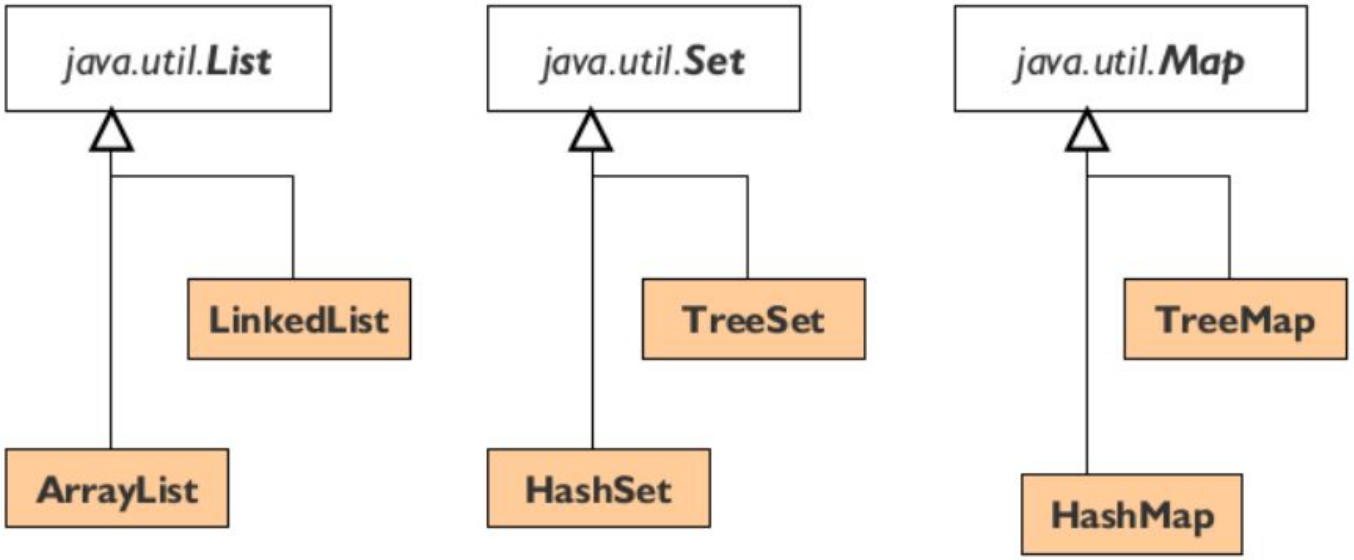
- TypeScript fornece uma API robusta para manipulação de coleções.
- Nas linguagens de mais baixo nível, podemos colecionar os dados em:
  - Vetores simples (estáticos), os quais representam um mecanismo nativo para colecionar dados, sejam eles objetos ou tipos primitivos.
  - Coleções, os quais representam conjuntos de classes robustas para a manipulação de estruturas de dados dinâmicas e complexas.
  - Em TypeScript praticamente só temos as coleções.

Vetores Estáticos

- Uma das formas mais eficientes de manter um conjunto de dados.
- Possuem tamanho fixo e não podem ser redimensionados.
- Seu tipo deve ser definido no momento da alocação de memória e deve ser compatível com sua referência.
- **Não temos vetores estáticos em TypeScript!**
  - No entanto, podemos simular vetores estáticos com Listas e Tuplas

Coleções do TypeScript

- Coleções compreendem um conjunto de interfaces e classes que implementam, listas, conjuntos e mapas de forma dinâmica.



Coleções do TypeScript

- IterableIterators
- Tuplas
- Arrays
- Sets
- Maps

IterableIterators

- Um objeto é considerado iterável se tiver uma implementação para a propriedade Symbol.iterator. Alguns tipos internos, como Array, Map, Set, String, Int32Array, Uint32Array, etc. têm sua propriedade Symbol.iterator já implementada. A função Symbol.iterator em um objeto é responsável por retornar a lista de valores para iterar.
- Com isso podemos utilizar um `for..of` ou um `for..in`.
- Não é tão usado pois, por si só, não tem muitas funções.
    - Mais utilizado em Arrays, Sets e Maps.

Tuplas

- Uma tupla funciona como um vetor estático, com a diferença que precisamos especificar o tipo de cada elemento.
- Tem tamanho limitado.
- Em TypeScript, são arrays.
  - Sendo assim, têm todas as funções de Arrays.
- Para uma tupla com ***n*** elementos, utilizamos a seguinte notação:
  - `let minhaTupla : [tipo1, tipo2, …, tipoN];`

Tuplas

```
let myTuple1 : [string, number, number, boolean];
let myTuple2 : [number, string] = [2018, "UFC"];

console.log(myTuple2[0]); // imprime 2018
console.log(myTuple2[1]); // imprime UFC

// o laço abaixo impreme os elementos da tupla, um por vez
// uma linha com 2018, outra com UFC
for(let elemento of myTuple2){
    console.log(elemento);
}
```

Arrays

- Em TypeScript, arrays (ou também chamados de listas), também funcionam como vetores (armazenamento e acesso a elementos), mas tem tamanho flexível.
  - Dois tipos de implementação: redimensionamento ou nós encadeados.
- Podemos inserir elementos de diferentes tipos, mas por padrão, cada array deve guardar elementos de um único tipo.
  - Chamamos de tipo base do Array.
- São eficientes para inserir e remover elementos.
- Não são eficientes para buscar elementos.

Arrays

- Como criar arrays de um tipo T qualquer em TypeScript?

```
let array1 : Array<T> = new Array<T>();
let array2 : Array<T> = [];
let array3 : T[] = [];
// criando arrays inicializados
let array4 : Array<T> = [ele1, ele2, ..., eleN];
let array5 : T[] = new Array<T>(ele1, ele2, ..., eleN);
// várias outras formas
```

Arrays

- Como acessar e modificar os elementos de arrays de um tipo T qualquer em TypeScript?
  - Utilizando o operador ***[ ]***, passando o índice do elemento.
  - Os índices (posições) dos arrays iniciam em zero.

```
let arr : Array<T> = [ele1, ele2, ..., eleN];

console.log(arr[4]); // acessa e imprime o quinto elemento
arr[2] = novoValor; // modifica o valor do terceiro elemento
```

Arrays

- Principais funções de arrays em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - *`push(valores : Array<T>) : number;`*
    - insere um conjunto de valores no fim do array e retorna o novo tamanho do array.
  - *`pop() : T;`*
    - remove o último elemento do array e o retorna.
  - *`shift() : T;`*
    - remove o primeiro elemento do array e o retorna.
  - *`unshift(valores : Array<T>) : number;`*
    - insere um conjunto de valores no início do array e retorna o novo tamanho do array.

Arrays

- Principais funções de arrays em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - *concat(outroArray : Array<T>) : Array<T>;*
    - cria um novo array com os valores do array que chamou a função mais os valores de outroArray, e retorna esse novo array.
  - *fill(valor : T, inicio : number, fim : number) : Array<T>*
    - Preenche o array com o valor, da posição início até a posição fim-1 e retorna o próprio array.
  - *indexOf(valor : T) : number;*
    - retorna o índice da primeira aparição de valor, ou -1 se o valor não estiver no array.
  - *lastIndexOf(valor : T) : number;*
    - retorna o índice da última aparição de valor, ou -1 se o valor não estiver no array.

Arrays

- Principais funções de arrays em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - *`join(separador : string = ",") : string;`*
    - retorna uma string com os valores do array separados pelo separador. Se o separador não for passado, a string separa os valores por vírgula.
  - *`reverse() : Array<T>;`*
    - coloca os elementos do array na ordem reversa, e retorna o próprio array.
  - *`slice(início : number, fim : number) : Array<T>;`*
    - retorna um novo array com os elementos do array original que vão do índice início até o índice fim-1, ou seja, uma “fatia” do array original.

Arrays

- Principais funções de arrays em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - *`sort(cmpFunction?) : Array<T>;`*
    - Ordena o array e retorna o próprio array.
    - Podemos passar a função de comparação, se quisermos.
  - *`entries() : IterableIterator<[number, T]>;`*
    - Retorna um IterableIterator de tuplas, em que cada tupla guarda o índice e o valor de um elemento do array.
  - *`keys() : IterableIterator<T>;`*
    - Retorna um IterableIterator com os índices dos elementos do array.
  - *`length : number;`*
    - Não é uma função, é um atributo.
    - Retorna o tamanho do array.

Sets e Maps

- Sets e Maps são coleções um pouco mais complexas, presentes apenas a partir do ECMA6.
- Assim, para utilizá-las, precisamos seguir alguns passos:
  - No nosso projeto (pasta dos arquivos de código) executamos o comando ***tsc --init***.
  - Isso criará um arquivo ***tsconfig.json***.
  - Abra esse arquivo.
  - À frente do campo ***"target":***, temos o valor ***"es5"***. Troque por ***"es6"***.
  - Salve o arquivo e feche-o.
  - Agora, podemos utilizar os Sets e os Maps.

Sets

- Sets (conjuntos), são um tipo especial de coleção.
- Nos sets, não é permitido inserir elementos repetidos.
- A própria estrutura deve garantir que não sejam inseridos elementos repetidos.
- Assim o programador não precisa se preocupar em verificar os elementos.
- Dependendo da implementação, não são muito eficientes para buscar, remover e inserir elementos.

Sets

- Como criar sets de um tipo T qualquer em TypeScript?

```
let mySet : Set<T> = new Set<T>();

mySet.add(ele1); // inserção funciona
mySet.add(ele2); // inserção funciona
mySet.add(ele3); // inserção funciona
mySet.add(ele1); // inserção não funciona
```

Sets

- Principais funções de sets em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - *add(valor : T) : Set<T>*
    - caso esse valor não esteja presente no seja, ele é inserido. É retornado o próprio set.
  - *delete(valor : T) : boolean;*
    - se o valor não estiver no set, é retornado falso, se o valor estiver no set, ele é removido e é retornado verdadeiro
  - *has(valor : T) : boolean;*
    - se o valor não estiver no set, é retornado falso, se o valor estiver no set, é retornado verdadeiro

Sets

- Principais funções de sets em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - `clear() : void;`
    - apaga todos os elementos do set e não retorna nada.
  - `values() : IterableIterator<T>;`
    - retorna um IterableIterator com todos os elementos no set.
  - `size : number;`
    - não é uma função, é um atributo.
    - retorna o tamanho do set.

Maps

- Maps (mapas), também são um tipo especial de coleção.
- Diferente das coleções vistas até agora, onde guardamos sempre elementos isolados, nos mapas são guardados pares.
- Os valores inseridos no mapa estão sempre associados a uma chave.
    - Por esse motivo são também chamados de arrays associativos.
    - Funcionam como dicionários.

Maps

- Os mapas têm como maior objetivo a recuperação rápida de elementos.
  - Em implementações como o HashMap, a busca e a inserção são muito eficientes.
- Quando inserimos um elemento, inserimos esse elemento relacionado a uma chave.
- Quando queremos buscar ou remover um elemento, o buscamos pela sua chave.
- Os valores do mapa podem se repetir, no entanto, as chaves não podem se repetir.
  - Cada chave está associada a um único valor.

Maps

- Como criar maps em que as chaves são de um tipo K qualquer e os valores são de um tipo V qualquer em TypeScript?

```
let myMap : Map<K,V> = new Map<K,V>();

myMap.set(chave1, valor1); // insere o valor 1
myMap.set(chave2, valor2); // insere o valor 2
myMap.set(chave1, valor3); // atualiza o valor relacionado a chave 1
```

Maps

- Principais funções de maps em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - `set(chave : K, valor : V) : Map<K,V>;`
    - se a chave já existir no map, atualiza o valor relacionado a chave, se a chave não existir, insere o valor relacionado a chave. É retornado o próprio map.
  - `get(chave : K) : V;`
    - se a chave existir no map, retorna o valor associado a chave, senão, retorna undefined.
  - `has(chave : K) : boolean;`
    - se a chave existir no map, retorna verdadeiro, senão, retorna falso.
  - `delete(chave : K) : boolean;`
    - se a chave existir no map, remove o elemento associado a chave e retorna verdadeiro, senão, retorna falso.

Maps

- Principais funções de maps em TypeScript:
  - podem ser utilizadas através do operador **.** (ponto)
  - *clear() : void;*
    - apaga todos os elementos do map e não retorna nada.
  - *entries() : IterableIterator<[K, V]>*
    - Retorna um IterableIterator de tuplas, em que cada tupla guarda a chave e o valor de um elemento do map.
  - *values() : IterableIterator<V>;*
    - Retorna um IterableIterator com os valores no map.
  - *keys() : IterableIterator<K>;*
    - Retorna um IterableIterator com as chaves do map.
  - *size : number;*
    - não é uma função, é um atributo.
    - retorna o número de elementos no map (uma chave e um valor conta como um).

Exercício

- Crie uma classe Funcionário que tem como atributos nome, cpf e salário.
- Crie um array com as informações de dez funcionários.
- Faça um pequeno programa para buscar funcionários nesse array pelo cpf.
  - O usuário digita o cpf do funcionário, o funcionário é buscado no array, e seu nome é impresso.
- Modifique o programa anterior para usar um mapa ao invés de um array.
  - Crie um mapa em que a chave são os cpfs do funcionários e o valor associado ao cpf é o próprio objeto funcionário.

Perguntas?