Exemples de Méthodes de Tableau et de Tests de Type en TypeScript

Ce document fournit des exemples clairs et concis pour chaque méthode de tableau et concept de test de type mentionné dans le diagramme.

1. Test de type union

1.1 Typeof

1.2 in

1.3 Instanceof

2. Méthodes de Tableau

2.1 IndexOf

2.2 ForEach

2.3 LastIndexOf

2.4 Every

2.5 Unshift

2.6 Sort

2.7 Splice

2.8 Filter

2.9 Shift

2.10 Reduce

2.11 Join

2.12 ReduceRight - ReduceLeft

2.13 Concat

2.14 Some

2.15 Pop

2.16 Number (probablement une erreur, ou se réfère à un concept de type Number)

1.1 Typeof

L'opérateur `typeof` est utilisé pour vérifier le type d'une variable primitive (string, number, boolean, symbol, bigint, undefined) ou pour déterminer si une variable est une fonction ou un objet.

Exemple :

```
function checkType(value: string | number | boolean) {
  if (typeof value === "string") {
    console.log(`C'est une chaîne: ${value.toUpperCase()}`);
  } else if (typeof value === "number") {
    console.log(`C'est un nombre: ${value * 2}`);
  } else {
    console.log(`C'est un booléen: ${!value}`);
  }
}

checkType("hello");   // Output: C'est une chaîne: HELLO
checkType(123);      // Output: C'est un nombre: 246
checkType(true);     // Output: C'est un booléen: false
```

1.2 in

L'opérateur `in` est utilisé pour vérifier si une propriété existe sur un objet. Il est utile pour affiner les types d'union basés sur la présence de propriétés différentes.

Exemple :

```typescript
interface Dog {
  bark(): void;
  name: string;
}

interface Cat {
  meow(): void;
  name: string;
}

type Animal = Dog | Cat;

function makeSound(animal: Animal) {
  if ("bark" in animal) {
    animal.bark(); // TypeScript sait que 'animal' est un Dog ici
  } else {
    animal.meow(); // TypeScript sait que 'animal' est un Cat ici
  }
}

const myDog: Dog = { bark: () => console.log("Woof!"), name: "Buddy" };
const myCat: Cat = { meow: () => console.log("Meow!"), name: "Whiskers" };

makeSound(myDog);  // Output: Woof!
makeSound(myCat);  // Output: Meow!
```

1.3 Instanceof

L'opérateur `instanceof` est utilisé pour vérifier si un objet est une instance d'une classe spécifique. Il est particulièrement utile pour affiner les types d'union lorsque les membres de l'union sont des instances de classes différentes.

Exemple :

```typescript
class Car {
  drive() { console.log("Driving a car"); }
}

class Bicycle {
  pedal() { console.log("Pedaling a bicycle"); }
}

type Vehicle = Car | Bicycle;

function startVehicle(vehicle: Vehicle) {
  if (vehicle instanceof Car) {
    vehicle.drive(); // TypeScript sait que 'vehicle' est une Car ici
  } else {
    vehicle.pedal(); // TypeScript sait que 'vehicle' est une Bicycle ici
  }
}

startVehicle(new Car());     // Output: Driving a car
startVehicle(new Bicycle()); // Output: Pedaling a bicycle
```

2. Méthodes de Tableau

2.1 IndexOf

La méthode `indexOf()` renvoie le premier index auquel un élément donné peut être trouvé dans le tableau, ou -1 s'il n'est pas présent.

Exemple :

```
const fruits = ["apple", "banana", "orange", "apple"];

console.log(fruits.indexOf("banana")); // Output: 1
console.log(fruits.indexOf("apple"));  // Output: 0 (première occurrence)
console.log(fruits.indexOf("grape"));  // Output: -1 (non trouvé)

// Recherche à partir d'un index spécifique
console.log(fruits.indexOf("apple", 1)); // Output: 3 (recherche à partir de
l'index 1)
```

2.2 ForEach

La méthode `forEach()` exécute une fonction fournie une fois pour chaque élément du tableau. Elle ne retourne rien (`void`).

Exemple :

```
const numbers = [1, 2, 3, 4, 5];
let sum = 0;

numbers.forEach(function(number) {
  sum += number;
});
console.log(sum); // Output: 15

// Avec une fonction fléchée
const fruits = ["apple", "banana", "cherry"];
fruits.forEach(fruit => console.log(`Je mange une ${fruit}.`));
// Output:
// Je mange une apple.
// Je mange une banana.
// Je mange une cherry.
```

2.3 LastIndexOf

La méthode `lastIndexOf()` renvoie le dernier index auquel un élément donné peut être trouvé dans le tableau, ou -1 s'il n'est pas présent. La recherche s'effectue à

rebours à partir de `fromIndex`.

Exemple :

```
const numbers = [2, 5, 9, 2, 8, 5];

console.log(numbers.lastIndexOf(2));  // Output: 3
console.log(numbers.lastIndexOf(5));  // Output: 5
console.log(numbers.lastIndexOf(9));  // Output: 2
console.log(numbers.lastIndexOf(7));  // Output: -1

// Recherche à partir d'un index spécifique (à rebours)
console.log(numbers.lastIndexOf(2, 3)); // Output: 3 (recherche 2 à partir de
l'index 3 vers le début)
console.log(numbers.lastIndexOf(5, 4)); // Output: 1 (recherche 5 à partir de
l'index 4 vers le début)
```

2.4 Every

La méthode `every()` teste si tous les éléments d'un tableau passent le test implémenté par la fonction fournie. Elle renvoie un booléen.

Exemple :

```
const numbers = [1, 2, 3, 4, 5];

const allPositive = numbers.every(num => num > 0);
console.log(allPositive); // Output: true

const allEven = numbers.every(num => num % 2 === 0);
console.log(allEven);     // Output: false

const emptyArray: number[] = [];
const emptyCheck = emptyArray.every(num => num > 0); // Toujours true pour un
tableau vide
console.log(emptyCheck); // Output: true
```

2.5 Unshift

La méthode `unshift()` ajoute un ou plusieurs éléments au début d'un tableau et renvoie la nouvelle longueur du tableau.

Exemple :

```
const fruits = ["banana", "orange"];
console.log(fruits.unshift("apple")); // Output: 3 (nouvelle longueur)
console.log(fruits);                  // Output: ["apple", "banana", "orange"]

fruits.unshift("grape", "kiwi");
console.log(fruits);                  // Output: ["grape", "kiwi", "apple",
"banana", "orange"]
```

2.6 Sort

La méthode `sort()` trie les éléments d'un tableau en place et renvoie le tableau trié. Le tri par défaut est alphabétique (conversion en chaîne de caractères) et croissant. Pour trier des nombres ou des objets, une fonction de comparaison est nécessaire.

Exemple :

```
const fruits = ["banana", "apple", "cherry"];
fruits.sort();
console.log(fruits); // Output: ["apple", "banana", "cherry"]

const numbers = [40, 1, 5, 200];
numbers.sort(); // Tri alphabétique par défaut
console.log(numbers); // Output: [1, 200, 40, 5]

// Tri numérique croissant
numbers.sort((a, b) => a - b);
console.log(numbers); // Output: [1, 5, 40, 200]

// Tri numérique décroissant
numbers.sort((a, b) => b - a);
console.log(numbers); // Output: [200, 40, 5, 1]

interface Product {
  name: string;
  price: number;
}

const products: Product[] = [
  { name: "Laptop", price: 1200 },
  { name: "Mouse", price: 25 },
  { name: "Keyboard", price: 75 }
];

// Tri par prix croissant
products.sort((a, b) => a.price - b.price);
console.log(products);
// Output: [
//   { name: 'Mouse', price: 25 },
//   { name: 'Keyboard', price: 75 },
//   { name: 'Laptop', price: 1200 }
// ]
```

2.7 Splice

La méthode `splice()` modifie le contenu d'un tableau en supprimant ou en remplaçant des éléments existants et/ou en ajoutant de nouveaux éléments en place. Elle renvoie un tableau contenant les éléments supprimés.

Exemple :

```
const fruits = ["apple", "banana", "cherry", "date", "elderberry"];

// Supprimer 1 élément à partir de l'index 2
let removed = fruits.splice(2, 1);
console.log(fruits);  // Output: ["apple", "banana", "date", "elderberry"]
console.log(removed); // Output: ["cherry"]

// Supprimer 2 éléments à partir de l'index 1 et ajouter de nouveaux éléments
removed = fruits.splice(1, 2, "grape", "kiwi");
console.log(fruits);  // Output: ["apple", "grape", "kiwi", "elderberry"]
console.log(removed); // Output: ["banana", "date"]

// Ajouter des éléments sans en supprimer (splice(index, 0, ...))
fruits.splice(1, 0, "lemon");
console.log(fruits);  // Output: ["apple", "lemon", "grape", "kiwi",
"elderberry"]
```

2.8 Filter

La méthode `filter()` crée un nouveau tableau avec tous les éléments qui passent le test implémenté par la fonction fournie. Elle ne modifie pas le tableau original.

Exemple :

```
const numbers = [1, 2, 3, 4, 5, 6];

const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // Output: [2, 4, 6]
console.log(numbers);     // Output: [1, 2, 3, 4, 5, 6] (original non modifié)

const words = ["spray", "limit", "elite", "exuberant", "present"];
const longWords = words.filter(word => word.length > 5);
console.log(longWords);   // Output: ["exuberant", "present"]
```

2.9 Shift

La méthode `shift()` supprime le **premier** élément d'un tableau et renvoie cet élément. Cette méthode modifie la longueur du tableau.

Exemple :

```
const fruits = ["apple", "banana", "cherry"];

const firstFruit = fruits.shift();
console.log(firstFruit); // Output: "apple"
console.log(fruits);     // Output: ["banana", "cherry"]

const emptyArray: string[] = [];
const shiftedEmpty = emptyArray.shift();
console.log(shiftedEmpty); // Output: undefined
console.log(emptyArray);   // Output: []
```

2.10 Reduce

La méthode `reduce()` exécute une fonction de rappel (reducer) sur chaque élément du tableau, de gauche à droite, afin de réduire le tableau à une seule valeur. La fonction de rappel prend un accumulateur et la valeur courante.

Exemple :

```
const numbers = [1, 2, 3, 4, 5];

// Somme de tous les nombres
const sum = numbers.reduce((accumulator, currentValue) => accumulator +
currentValue, 0);
console.log(sum); // Output: 15

// Concaténation de chaînes
const words = ["Hello", "World", "TypeScript"];
const sentence = words.reduce((acc, word) => acc + " " + word);
console.log(sentence); // Output: "Hello World TypeScript"

// Compter les occurrences d'éléments
const fruits = ["apple", "banana", "apple", "orange", "banana", "apple"];
const fruitCount = fruits.reduce((acc: { [key: string]: number }, fruit) => {
  acc[fruit] = (acc[fruit] || 0) + 1;
  return acc;
}, {});
console.log(fruitCount); // Output: { apple: 3, banana: 2, orange: 1 }
```

2.11 Join

La méthode `join()` crée et renvoie une nouvelle chaîne de caractères en concaténant tous les éléments d'un tableau. La concaténation est effectuée en utilisant un séparateur spécifié (par défaut, une virgule).

Exemple :

```
const elements = ['Fire', 'Air', 'Water'];

console.log(elements.join());     // Output: "Fire,Air,Water"
console.log(elements.join(''));   // Output: "FireAirWater"
console.log(elements.join('-'));  // Output: "Fire-Air-Water"

const numbers = [1, 2, 3];
console.log(numbers.join(' + ')); // Output: "1 + 2 + 3"
```

2.12 ReduceRight - ReduceLeft

- **`reduce()` (ou `reduceLeft`)** : Exécute une fonction de rappel sur chaque élément du tableau, de gauche à droite, afin de réduire le tableau à une seule valeur. C'est la même chose que `reduce()`.
- **`reduceRight()`** : Exécute une fonction de rappel sur chaque élément du tableau, de droite à gauche, afin de réduire le tableau à une seule valeur.

Exemple :

```
const numbers = [1, 2, 3, 4];

// reduce (de gauche à droite)
const sumLeft = numbers.reduce((acc, val) => acc + val, 0);
console.log(sumLeft); // Output: 10

// reduceRight (de droite à gauche)
const flattened = [[0, 1], [2, 3], [4, 5]].reduceRight((acc, val) =>
acc.concat(val), []);
console.log(flattened); // Output: [4, 5, 2, 3, 0, 1]

const words = ["a", "b", "c"];
const resultLeft = words.reduce((acc, val) => acc + val, "");
console.log(resultLeft); // Output: "abc"

const resultRight = words.reduceRight((acc, val) => acc + val, "");
console.log(resultRight); // Output: "cba"
```

2.13 Concat

La méthode `concat()` est utilisée pour fusionner deux ou plusieurs tableaux. Cette méthode ne modifie pas les tableaux existants, mais renvoie un nouveau tableau.

Exemple :

```
const arr1 = [1, 2];
const arr2 = [3, 4];
const arr3 = [5, 6];

const newArray = arr1.concat(arr2, arr3);
console.log(newArray); // Output: [1, 2, 3, 4, 5, 6]

const fruits = ["apple", "banana"];
const moreFruits = ["cherry", "date"];
const allFruits = fruits.concat(moreFruits);
console.log(allFruits); // Output: ["apple", "banana", "cherry", "date"]

// Concaténation avec des valeurs individuelles
const combined = arr1.concat(3, [4, 5]);
console.log(combined); // Output: [1, 2, 3, 4, 5]
```

2.14 Some

La méthode `some()` teste si au moins un élément du tableau passe le test implémenté par la fonction fournie. Elle renvoie un booléen.

Exemple :

```
const numbers = [1, 2, 3, 4, 5];

const hasEven = numbers.some(num => num % 2 === 0);
console.log(hasEven); // Output: true (2 et 4 sont pairs)

const hasNegative = numbers.some(num => num < 0);
console.log(hasNegative); // Output: false

const words = ["apple", "banana", "cat", "dog"];
const hasLongWord = words.some(word => word.length > 6);
console.log(hasLongWord); // Output: true ("banana" est plus long que 6)
```

2.15 Pop

La méthode `pop()` supprime le **dernier** élément d'un tableau et renvoie cet élément. Cette méthode modifie la longueur du tableau.

Exemple :

```
const fruits = ["apple", "banana", "cherry"];

const lastFruit = fruits.pop();
console.log(lastFruit); // Output: "cherry"
console.log(fruits);    // Output: ["apple", "banana"]

const emptyArray: string[] = [];
const poppedEmpty = emptyArray.pop();
console.log(poppedEmpty); // Output: undefined
console.log(emptyArray);  // Output: []
```

2.16 Number

Le terme "Number" dans ce contexte, bien que n'étant pas une méthode de tableau à proprement parler, fait probablement référence à la conversion de valeurs en type numérique ou à l'objet global `Number` en JavaScript/TypeScript.

Le constructeur `Number()` peut être utilisé pour convertir n'importe quelle valeur en un nombre. Si la valeur ne peut pas être convertie, elle renvoie `NaN` (Not-a-Number).

Exemple : Conversion de type en nombre

```
// Conversion de chaînes de caractères en nombres
console.log(Number("123"));     // Output: 123
console.log(Number("3.14"));    // Output: 3.14
console.log(Number("  10  "));  // Output: 10 (les espaces sont ignorés)
console.log(Number("abc"));     // Output: NaN
console.log(Number(""));        // Output: 0 (chaîne vide)

// Conversion de booléens en nombres
console.log(Number(true));      // Output: 1
console.log(Number(false));     // Output: 0

// Conversion de null et undefined
console.log(Number(null));      // Output: 0
console.log(Number(undefined)); // Output: NaN

// Utilisation avec des opérateurs unaires (+)
console.log(+"456");           // Output: 456
console.log(+"hello");         // Output: NaN
```

Il est également important de se rappeler que `number` est le type primitif en TypeScript pour les valeurs numériques.

```
let age: number = 30;
let price: number = 99.99;
// let invalid: number = "text"; // Erreur: Type 'string' n'est pas assignable
au type 'number'.
```