Exemples Supplémentaires de Signatures d'Index (Index Signatures) en TypeScript

Reprenons le concept avec quelques cas d'utilisation pratiques et des pièges courants à surveiller.

1. Modéliser des Réponses d'API Dynamiques

C'est un scénario très courant. Imaginez que vous interrogez une API qui renvoie des données de configuration ou des métriques où les noms des clés peuvent varier.

Scénario : Une API renvoie les statistiques d'utilisation par pays. Les noms des pays sont les clés, et la valeur est un objet contenant `utilisateurs` et `revenus`.

```
interface StatistiquesPays {
  utilisateurs: number;
  revenus: number;
}

interface DonneesGlobales {
  // La signature d'index indique que toute clé string sera mappée à un objet Statisti
  [nomDuPays: string]: StatistiquesPays;
  // On peut aussi avoir des propriétés fixes si besoin
  derniereMiseAJour: string;
}

const donneesRecues: DonneesGlobales = {
  France: {
    utilisateurs: 1500000,
    revenus: 5000000
  },
  Allemagne: {
    utilisateurs: 1200000,
    revenus: 4500000
  },
  // La signature d'index permet d'ajouter n'importe quel autre pays
  Espagne: {
    utilisateurs: 800000,
    revenus: 3000000
  },
  dernieremiseAJour: "2024-07-30T10:30:00Z"
};

console.log(donneesRecues.France.utilisateurs);      // Output: 1500000
console.log(donneesRecues['Allemagne'].revenus);    // Output: 4500000
console.log(donneesRecues.dernieremiseAJour); // Output: 2024-07-30T10:30:00Z
```

```
// TypeScript vous permet d'accéder à des clés dynamiques sans erreur de compilation
const paysRecherche = 'Italie';
// Note: Le type de 'donneesRecues[paysRecherche]' est 'StatistiquesPays' (ou 'string'
const statsItalie = donneesRecues[paysRecherche];

if (statsItalie) {
  console.log(`Utilisateurs en ${paysRecherche}: ${statsItalie.utilisateurs}`);
} else {
  console.log(`${paysRecherche} non trouvé.`); // Output: Italie non trouvé.
}

// Erreur possible si vous essayez d'assigner une valeur incorrecte
/*
donneesRecues.Canada = {
  utilisateurs: 1000000,
  revenus: 'cinq millions' // Erreur TS: Type 'string' n'est pas assignable au type 'n
};
*/
```

Explication : La signature `[nomDuPays: string]: StatistiquesPays;` permet d'avoir un nombre illimité de propriétés dont le nom est une `string`, et dont la valeur respecte l'interface `StatistiquesPays`. La propriété `derniereMiseAJour` est une propriété nommée et fixe.

2. Dictionnaire de Configurations Flexibles

Scénario : Votre application a différentes options de configuration, et certaines d'entre elles peuvent être de types variés (chaînes, nombres, booléens).

```
type ConfigurationValue = string | number | boolean;

interface AppConfig {
  [key: string]: ConfigurationValue; // Toute clé string peut avoir une valeur string,
  appName: string; // Propriété nommée, son type (string) est compatible avec Configur
  version: number; // Propriété nommée, son type (number) est compatible avec Configur
}

const config: AppConfig = {
  appName: 'MonSuperApp',
  version: 1.2,
  debugMode: true,
  theme: 'dark',
  maxUsers: 1000,
  // initialLoadTime: new Date() // Erreur TS: Type 'Date' n'est pas assignable au typ
};

console.log(config.appName);    // Output: MonSuperApp
console.log(config.debugMode);  // Output: true
```

```
console.log(config['maxUsers']); // 

// On peut itérer sur les clés de manière sécurisée (si on est sûr qu'elles existent)
for (const key in config) {
    if (Object.prototype.hasOwnProperty.call(config, key)) {
        console.log(`${key}: ${config[key]}`); // TypeScript sait que config[key] est
    }
}
/* Output:
appName: MonSuperApp
version: 1.2
debugMode: true
theme: dark
maxUsers: 1000
*/
```

Explication : L'utilisation d'une union de types (`string | number | boolean`) pour la valeur de la signature d'index permet une grande flexibilité. Les propriétés nommées `appName` et `version` doivent simplement s'aligner avec l'un des types de cette union.

3. Comprendre la Limitation : Une Seule Signature `string`

Scénario : Vous voulez un objet où certaines clés donnent des nombres et d'autres des chaînes. Vous pourriez penser à créer deux signatures d'index de type `string` .

```
// Ceci ne fonctionnera PAS en TypeScript !
/*
interface MultiTypedObject {
  [key: string]: number; // Signature 1: clés string -> number
  [key: string]: string; // Signature 2: clés string -> string
}
*/
```

Pourquoi cette erreur ? TypeScript ne peut pas savoir quel type utiliser si vous accédez à une propriété via `myObject['someKey']` . La clé `'someKey'` est une chaîne. Devrait-elle retourner un `number` ou un `string` ? Pour éviter cette ambiguïté, TypeScript l'interdit.

La solution est d'utiliser une union de types pour la valeur de la signature d'index :

```
interface MixedData {
  [key: string]: number | string | boolean; // Une seule signature, avec une union de
  // Vous pouvez toujours avoir des propriétés nommées qui respectent cette union
  id: number;
  name: string;
  isActive: boolean;
}
```

```
const mixed: MixedData = {
  id: 1,
  name: 'John',
  isActive: true,
  dynamicKey1: 123,
  dynamicKey2: 'value',
  dynamicKey3: false
};

console.log(mixed.id);          // 1 (TypeScript sait que c'est un nombre grâce à la p
console.log(mixed.dynamicKey1); // 123 (TypeScript sait que c'est number | string | bo

// Vous devrez peut-être faire des vérifications de type lors de l'accès à des proprié
const value = mixed['dynamicKey1'];
if (typeof value === 'number') {
    console.log(`Ceci est un nombre: ${value}`); // Output: Ceci est un nombre: 123
}
```

4. Propriétés Nommées DOIVENT être compatibles

Scénario : Vous avez une signature d'index, mais vous définissez aussi des propriétés nommées qui ne respectent pas le type de la signature.

```
interface Inventaire {
  [itemName: string]: number; // Toute clé string doit avoir une valeur de type number
  totalItems: number;          // OK: 'number' est compatible avec 'number'
  // description: string;    // ERREUR TS: 'string' n'est pas assignable à 'number'.
                             // La propriété nommée 'description' de type string
                             // entre en conflit avec la signature d'index string -> n
}

const stock: Inventaire = {
  'Apple': 50,
  'Banana': 20,
  totalItems: 70
};

// stock.description = "Fruit stock"; // Ne compilera pas
```

Explication : TypeScript garantit que si vous accédez à `Inventaire['Apple']` (qui est une `string` en tant que clé), le type de retour est `number` (selon la signature d'index). Si `description` était une `string`, alors `Inventaire['description']` serait `string`, ce qui contredirait la signature d'index. C'est pourquoi toutes les propriétés nommées doivent être un sous-type ou le même type que la valeur de la signature d'index.

Ces exemples supplémentaires devraient solidifier votre compréhension des signatures d'index et de leur utilisation pratique en TypeScript !