Tipos de Acesso Indexado

Podemos usar um tipo de acesso indexado para buscar uma propriedade específica em outro tipo:

```
type Person = { age: number; name: string; alive: boolean };
type Age = Person["age"];

type Age = number
```

O tipo de indexação é, ele mesmo, um tipo, então podemos usar uniões, `keyof` ou outros tipos inteiros:

```
type I1 = Person["age" | "name"];

type I1 = string | number

type I2 = Person[keyof Person];

type I2 = string | number | boolean

type AliveOrName = "alive" | "name";
type I3 = Person[AliveOrName];

type I3 = string | boolean
```

Você verá um erro se tentar indexar uma propriedade que não existe:

```
type I1 = Person["alve"];
Property 'alve' does not exist on type 'Person'.
```

Outro exemplo de indexação com um tipo arbitrário é usar `number` para obter o tipo dos elementos de um array. Podemos combinar isso com `typeof` para capturar convenientemente o tipo do elemento de um literal de array:

```
const MyArray = [
  { name: "Alice", age: 15 },
  { name: "Bob", age: 23 },
  { name: "Eve", age: 38 },
];

type Person = typeof MyArray[number];

type Person = {
    name: string;
    age: number;
}
type Age = typeof MyArray[number]["age"];

type Age = number
// Ou
type Age2 = Person["age"];

type Age2 = number
```

Você só pode usar tipos ao indexar, o que significa que não pode usar um `const` para fazer uma referência de variável:

```
const key = "age";
type Age = Person[key];
Type 'key' cannot be used as an index type.
'key' refers to a value, but is being used as a type here. Did you mean 'typeof key'?
```

No entanto, você pode usar um alias de tipo para um estilo de refatoração semelhante:

```
type key = "age";
type Age = Person[key];
```