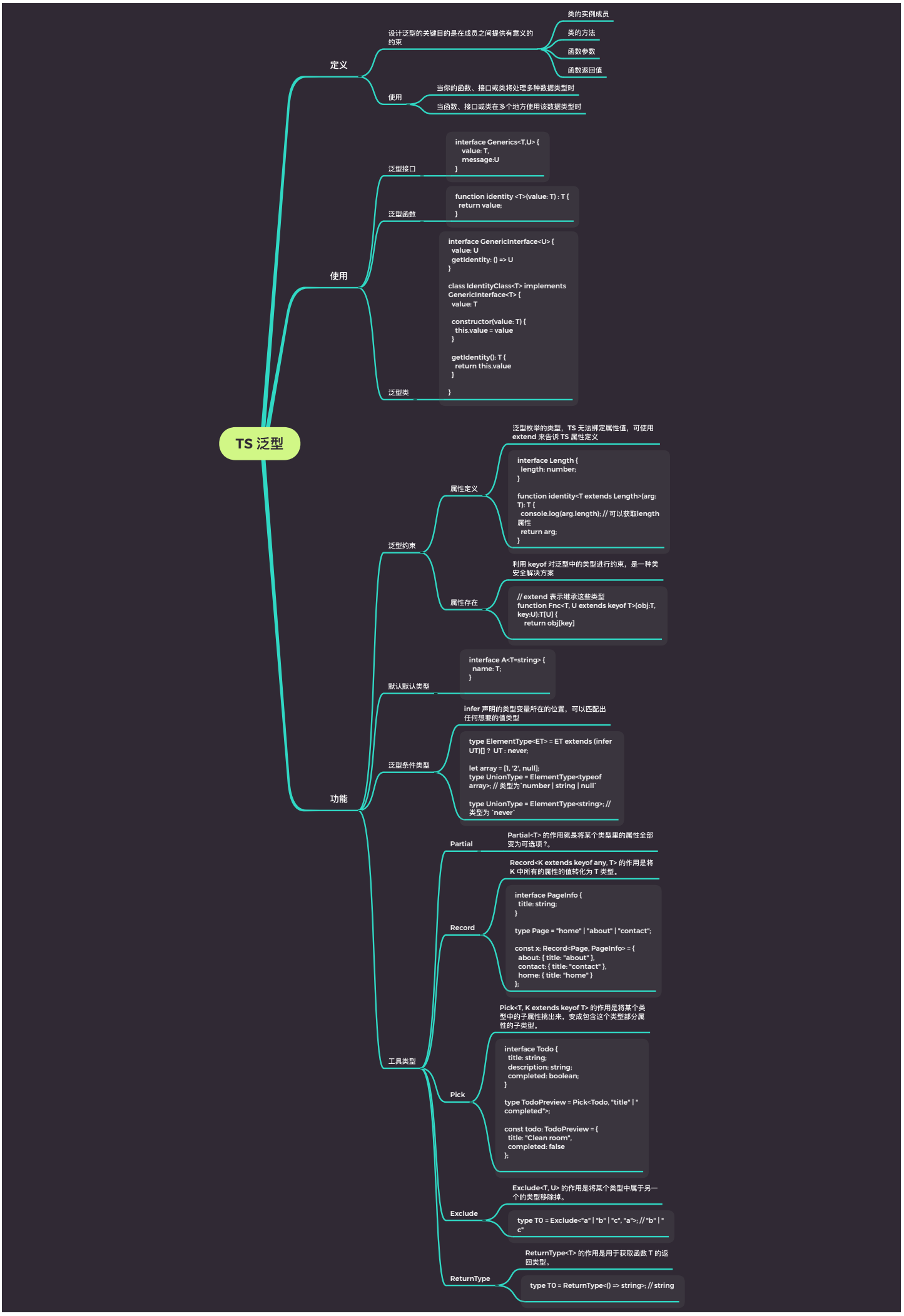
TS 泛型
定义
设计泛型的关键目的是在成员之间提供有意义的约束
类的实例成员
类的方法
函数参数
函数返回值
使用
当你的函数、接口或类将处理多种数据类型时
当函数、接口或类在多个地方使用该数据类型时
使用
泛型接口
interface Generics<T,U> {
value: T,
message:U
}
泛型函数
function identity <T>(value: T) : T {
return value;
}
泛型类
interface GenericInterface<U> {
value: U
getIdentity: () => U
}
class IdentityClass<T> implements GenericInterface<T> {
value: T
constructor(value: T) {
this.value = value
}
getIdentity(): T {
return this.value
}
}
功能
泛型约束
属性定义
泛型枚举的类型，TS 无法绑定属性值，可使用 extend 来告诉 TS 属性定义
interface Length {
length: number;
}
function identity<T extends Length>(arg: T): T {
console.log(arg.length); // 可以获取length 属性
return arg;
}
属性存在
利用 keyof 对泛型中的类型进行约束，是一种类安全解决方案
// extend 表示继承这些类型
function Fnc<T, U extends keyof T>(obj:T, key:U):T[U] {
return obj[key]
}
默认默认类型
interface A<T=string> {
name: T;
}
泛型条件类型
infer 声明的类型变量所在的位置，可以匹配出任何想要的值类型
type ElementType<ET> = ET extends (infer UT)[] ? UT : never;
let array = [1, '2', null];
type UnionType = ElementType<typeof array>; // 类型为`number | string | null`
type UnionType = ElementType<string>; // 类型为 `never`
工具类型
Partial
Partial<T> 的作用就是将某个类型里的属性全部变为可选项 ?。
Record
Record<K extends keyof any, T> 的作用是将 K 中所有的属性的值转化为 T 类型。
interface PageInfo {
title: string;
}
type Page = "home" | "about" | "contact";
const x: Record<Page, PageInfo> = {
about: { title: "about" },
contact: { title: "contact" },
home: { title: "home" }
};
Pick
Pick<T, K extends keyof T> 的作用是将某个类型中的子属性挑出来，变成包含这个类型部分属性的子类型。
interface Todo {
title: string;
description: string;
completed: boolean;
}
type TodoPreview = Pick<Todo, "title" | "completed">;
const todo: TodoPreview = {
title: "Clean room",
completed: false
};
Exclude
Exclude<T, U> 的作用是将某个类型中属于另一个的类型移除掉。
type T0 = Exclude<"a" | "b" | "c", "a">; // "b" | "c"
ReturnType
ReturnType<T> 的作用是用于获取函数 T 的返回类型。
type T0 = ReturnType<() => string>; // string