تایپ اسکریپت دسته گسترده ای از Utility Type ها برای ما فراهم می کند که جهت سهولت کار درانجام برخی عملیات به کار میروند. این تایپ ها قابل پیاده سازی به صورت دستی هستند و وجودشان صرفا انجام کاری را آسان تر می کند نه ممکن. بیشتر این تایپ ها برای انجام عملیات و تغییرات روی تایپ های پیچیده تر مانند UnionType ها یا تایپ های حاوی پراپرتی به کار میروند.

انواع تایپ هایی که بیشترین کارایی را دارند:

تایپ Awaited<Type> :

از آنجایی که تایپ Promise یک تایپ تعریف شده در تایپ اسکریپت است (type A = Promise<string>; به ما Promise برمیگرداند ) برای یافتن تایپ بعد از محقق شدن Promise ها از Awaited استفاده میکنیم. به طوری که type A = Awaited<Promise<string>>; به ما string برمیگرداند نه Promise .

تایپ Partial<Type> :

اگر بخواهیم تایپ تمام پراپرتی های یک تایپ را به حالت optional تغییر دهیم، باید در آن تایپ پیمایش کرده و یکی یکی این تایپ ها را تغییر دهیم. Partial به ما امکان انجام بسیار ساده تر این تغییر را می دهد.

تایپ Required<Type> :

برعکس Partial، تایپ تمام پراپرتی های یک تایپ را (در صورت optional بودن) به حالت required تبدیل می کند.

تایپ Readonly<Type> :

اگر کلید readonly در ابتدای تایپی بیاید آن تایپ قابل تغییر نخواهد بود. تایپ Readonly مانند تایپ های قبلی در تمامی پراپرتی های یک تایپ پیمایش میکند و تایپ آنها را readonly میکند.

تایپ Record<Keys, Type> :

تایپی میسازد با key های فراهم شده در قسمت اول ورودی و valueهایی از جنس تایپ فراهم شده در قسمت دوم ورودی. برای map کردن به کار میرود.

تایپ Pick<Type, Keys> :

تایپی را به عنوان ورودی اول خود پاس گرفته و تایپی میسازد حاوی پراپرتی هایی مبتنی بر تایپ اصلی دارای کلید های فراهم شده در ورودی دوم خود. در واقع Pick زیرمجموعه ای از تایپ ورودی خود شامل کلید های فراهم شده میسازد.

مثال:

interface Todo {

title: string;

description: string;

completed: boolean;

}

type TodoPreview = Pick<Todo, "title" | "completed">;

const todo: TodoPreview = {

title: "Clean room",

completed: false,

};

تایپ Omit<Type, Keys> :

برعکس تایپ Pick، این تایپ زیرمجموعه ای از تایپ ورودی خود میسازد که فاقد کلید های فراهم شده در ورودی دوم آن است (به نوعی مانند یک فیلتر عمل میکند).

مثال:

interface Todo {

title: string;

description: string;

completed: boolean;

createdAt: number;

}

type TodoPreview = Omit<Todo, "description">;

const todo: TodoPreview = {

title: "Clean room",

completed: false,

createdAt: 1615544252770,

};

تایپ Exclude<UnionType, ExcludedMembers> :

این تایپ یک UnionType به عنوان ورودی خود میگیرد (برای مثال به شکل "a" | "b" | "c" ) و یک یا چند تایپ (در صورت بیشتر از یکی بودن بازهم به شکل UnionType) میگیرد و یک تایپ یا UnionType از ورودی اول خود میسازد که فاقد تایپ های ورودی دوم خود است (این هم به عنوان نوعی فیلتر برای تایپ ها به کار می رود).

تایپ Extract<Type, Union> :

این تایپ برعکس تایپ Exclude، تایپی از ورودی اول خود می سازد که در ورودی دوم آن نیز موجود باشد. در واقع بین دو تایپ ورودی خود اشتراک می گیرد و تایپ های مشترک بین آن دو را برمیگرداند.

تایپ NonNullable<Type> :

در صورت وجود تایپ های null و undefined در UnionType فراهم شده، آن ها را حذف میکند.

تایپ Parameters<Type> :

تایپی می سازد از جنس پارامتر(آرگومان) ورودی تایپ تابع پاس داده شده به آن. مثال:

type T1 = Parameters<(s: string) => void>;

type T1 = [s: string]

تایپ ReturnType<Type> :

برعکس تایپ Parameters این تایپ، تایپی می سازد از جنس خروجی (return) تایپ تابع پاس داده شده به آن.

مثال:

type T0 = ReturnType<() => string>;

type T0 = string

type T1 = ReturnType<(s: string) => void>;

type T1 = void

تایپ NoInfer<Type> :

در تایپ اسکریپت در صورت فراهم نشدن تایپی به صورت پیش فرض، تایپ مقدار assign شده برای آن تایپ فرض میشود که به این عمل inference می گویند. تایپ NoInfer جلوی این عمل را میگیرد.

مثال:

function createStreetLight<C extends string>(

colors: C[],

defaultColor?: NoInfer<C>,

) {

// ...

}

createStreetLight(["red", "yellow", "green"], "red"); // OK

createStreetLight(["red", "yellow", "green"], "blue"); // Error