



**Code
Academy**



Tipų formavimas ir susaistymas

TypeScript

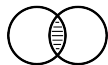
5 paskaita



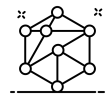
Paskaitos eiga



Tipų indeksai ir `keyof` operatorius



Tipų apjungimas ir `&` sankirtos operatorius



Tipų susaistymas



Pagalbiniai tipai

Tipų indeksai ir keyof operatorius





Tipų indeksai

Aprašius objektų tipus, galime gauti kokios savybės tipą rašydami tokią sintaksę→

`Tipas["savybėsPavadinimas"]`

Praktikoje ši sintaksė naudojama:

- Naudojant sudėtingus tipus, kurie buvo kuriami pagal bendrinius(generic) kintamuosius
- Iteruojant per tipo savybes kitų tipų susaistymo metu

```
interface Person {  
  id: string | number;  
  fullname: string;  
  age: number;  
};
```

```
type PersonId = string | number  
type PersonId = Person['id'];
```

```
type PersonFullname = string  
type PersonFullname = Person['fullname'];
```

```
type PersonAge = number  
type PersonAge = Person['age'];
```



Keyof operatorius

Keyof operatorius skirtas gauti, objekto tipo rakto tipą.

Keyof operatorius naudojamas praktikoje:

- Dinamiškai formuojant ir konstruojant tipus
- Dinamiškai formuojant ir konstruojant reikšmes

```
interface Person {  
  id: string | number;  
  fullname: string;  
  age: number;  
};
```

```
type PersonKey = keyof Person;
```

```
const personKey1: PersonKey = "age";
```

```
const personKey2: PersonKey = "id";
```

```
const personKey3: PersonKey = "fullname";
```

```
const personKey4: keyof Person
```

Type '"neteisingas"' is not assignable to type 'keyof Person'. ts(2322)

[View Problem](#) No quick fixes available

```
const personKey4: PersonKey = "neteisingas";
```



Keyof ir tipų indeksų pavyzdžiai kuriant dinamiškus tipus

```
interface Car {  
  id: number;  
  model: string;  
  makeYear: number;  
};  
  
type ReplaceObjectPropTypes<ObjectType, ToReplace, ReplaceWith> = {  
  [Key in keyof ObjectType]: ObjectType[Key] extends ToReplace ? ReplaceWith : ObjectType[Key];  
}  
  
type CarWithStringifiedNumberProps = {  
  id: string;  
  model: string;  
  makeYear: string;  
}  
  
type CarWithStringifiedNumberProps = ReplaceObjectPropTypes<Car, number, string>;
```

Klausimai?



Tipų apjungimas ir “&” sankirtos operatorius





Tipų apjungimas ir “&” sankirtos operatorius

Kartais reikia sukurti objekto tipą, kito objekto tipo pagrindu. Tokiu atveju būtų korektiška perpanaudoti jau esantį pagrindo tipą, kad papildytas tipas visada su juo sutaptų.

type alias apjungimas

```
type Person = {  
  name: string,  
  surname: string,  
}  
  
type Student = Person & {  
  course: number,  
  marks: number[],  
}
```

```
const stud: Student;
```

stud.

- course
- marks
- name
- surname

interface apjungimas

```
interface Person {  
  name: string;  
  surname: string;  
}  
  
You, 12 seconds ago | 1 author (You)  
interface Student extends Person {  
  course: number,  
  marks: number[],  
}
```

```
const stud: Student;
```

stud.

- course
- marks
- name
- surname

Klausimai?



Tipų susaistymas





Tipų susaistymas

Tipų susaistymas (type mapping) - tai naujų tipų kūrimas naudojant jau egzistuojančius tipus.

Naujų tipų saistymas atliekamas iteruojant per egzistuojančio tipo savybes ir formuojant naujai kuriamo (saistomo) objekto savybių tipus.

Taip daroma, dėl vienos pagrindinių programavimo paradigmos - DRY (Don't Repeat Yourself).

Pritaikius naujų tipų kūrimui saistymo metodus įgauname gerąją praktiką - pasikeitus pagrindiniams tipams, automatiškai pasikeičia ir susaistomi tipai.



Inkapsuluoto tipo generavimas pagal egzistuojantį tipą

```
type Accommodation = {  
  address: string,  
  squares: number,  
  type: 'Flat' | 'House' | 'Cottage',  
}  
  
type AccommodationGetters = {  
  getAddress: () => string;  
  getSquares: () => number;  
  getType: () => "Flat" | "House" | "Cottage";  
}  
  
type AccommodationSetters = {  
  setAddress: (val: string) => void;  
  setSquares: (val: number) => void;  
  setType: (val: "Flat" | "House" | "Cottage") => void;  
}  
  
type AccommodationGetters & AccommodationSetters = {  
  [Key in keyof Accommodation as `get${Capitalize<Key>}`]: () => Accommodation[Key]  
}  
  
type AccommodationSetters & AccommodationGetters = {  
  [Key in keyof Accommodation as `set${Capitalize<Key>}`]: (val: Accommodation[Key]) => void  
}
```

```
type AccommodationIncapsulated =  
  AccommodationGetters & AccommodationSetters;  
  
let accomodation: AccommodationIncapsulated;  
accomodation.  
  getAddress  
  getSquares  
  getType  
  setAddress  
  setSquares  
  setType
```



Objekto savybių išrinkimas pagal tipą

```
type MultipleColor = {  
  main: string,  
  light: string,  
  dark: string  
};
```

```
type Palette = {  
  primary: string,  
  secondary: string,  
  success: MultipleColor,  
  danger: MultipleColor,  
};
```

```
type PaletteMultipleColorKeyObject = {  
  primary: never;  
  secondary: never;  
  success: "success";  
  danger: "danger";  
}
```

```
type PaletteMultipleColorKeyObject = {  
  [Key in keyof Palette]: Palette[Key] extends MultipleColor ? Key : never  
};
```

```
type PaletteMultipleColorKeys = "success" | "danger"
```

```
type PaletteMultipleColorKeys = PaletteMultipleColorKeyObject[keyof Palette];
```

```
type PaletteMultipleColors = {  
  success: MultipleColor;  
  danger: MultipleColor;  
}
```

```
type PaletteMultipleColors = Pick<Palette, PaletteMultipleColorKeys>;
```

Klausimai?



Pagalbiniai tipai





Pagalbiniai tipai

Pagalbiniai tipai, tai TypeScript bibliotekos iš anksto aprašyti tipai, kurie susaisto naujus tipus pagal perduotus bendrinius tipus.

Bendriniai tipai, lyg funkcijos - yra perpanaudojami ir priima kitus tipus. Tuomet bendrinio tipo aprašyme galime dinamiškai kurti tipus, daugiau - kitą pamoką).



Pagalbiniai tipai

Dažniausiai naudojami pagalbiniai tipai:

`Partial<Type>` - visos Type savybės tampa neprivalomomis

`Required<Type>` - visos Type savybės tampa privalomomis

`Record<Keys, Type>` - sudaromas objektas kur, sąjunga aprašytas raktais Keys, bus pasiekiamos Type tipo reikšmės

`Pick<Type, Keys>` - sudaromas objektas kur iš Type tipo pasirenkamos savybės apibūdintos raktų sąjunga Keys

`Omit<Type, Keys>` - sudaromas objektas kur iš Type tipo pašalinamos savybės apibūdintos raktų sąjunga Keys

`ReturnType<Type>` - grąžina funkcijos Type grąžinimo tipą



Pagalbinių tipų naudojimo pavyzdžiai

```
type User = {  
  email: string,  
  password: string,  
  name?: string,  
  surname?: string,  
  image?: string,  
  cartItems: string[],  
};  
  
type UserRegistration = Omit<User, 'cartItems'> & {  
  repeatEmail: User['email'],  
  repeatPassword: User['password'],  
};  
  
let a: UserRegistration;  
a.  
  You, 3 seconds ago • Uncommitted changes  
  email (property) email: string  
  image?  
  name?  
  password  
  repeatEmail  
  repeatPassword  
  surname?
```

```
type User = {  
  email: string,  
  password: string,  
  name?: string,  
  surname?: string,  
  image?: string,  
  cartItems: string[],  
};  
  
type UserUpdate = {  
  cartItems: string[];  
  name?: string | undefined;  
  surname?: string | undefined;  
  image?: string | undefined;  
}  
  
type UserUpdate = Omit<User, 'email' | 'password'> ;
```



Pagalbinių tipų naudojimo pavyzdžiai

```
type User = {  
  email: string,  
  password: string,  
  name?: string,  
  surname?: string,  
  image?: string,  
  cartItems: string[],  
};  
  
type UserOptionalProps = {  
  name?: string | undefined;  
  surname?: string | undefined;  
  image?: string | undefined;  
}  
  
type UserOptionalProps = Pick<User, 'name' | 'surname' | 'image'>;
```

```
type UserPropsRequired = {  
  name: string;  
  surname: string;  
  image: string;  
}  
  
type UserPropsRequired = Required<UserOptionalProps>;
```

```
type EssentialUsers = {  
  admin: User;  
  moderator: User;  
  manager: User;  
}  
  
type EssentialUsers = Record<  
  'admin' | 'moderator' | 'manager',  
  User  
>;
```



Tipų grupavimas pagal savybę

```
interface Car {  
  id: number;  
  model: string;  
  makeYear: number;  
};  
  
type CarsGroupedByModel = {  
  [Key: Car['model']]: Omit<Car, 'model'>[]  
}
```

```
const cars: Car[] = [  
  { id: 1, model: 'Citroen C4', makeYear: 2000 },  
  { id: 2, model: 'Citroen C5', makeYear: 2002 },  
  { id: 3, model: 'Citroen C6', makeYear: 2004 },  
  { id: 4, model: 'Citroen C4', makeYear: 2006 },  
  { id: 5, model: 'Citroen C5', makeYear: 2010 },  
];
```

```
const carsGroupedByModel = cars.reduce<CarsGroupedByModel>((prevGroups, { model, ...car }) => {  
  if (model in prevGroups) {  
    prevGroups[model].push(car);  
  } else {  
    prevGroups[model] = [car];  
  }  
  return prevGroups;  
}, {});
```

(index)	0	1
Citroen C4	{ id: 1, makeYear: 2000 }	{ id: 4, makeYear: 2006 }
Citroen C5	{ id: 2, makeYear: 2002 }	{ id: 5, makeYear: 2010 }
Citroen C6	{ id: 3, makeYear: 2004 }	

Klausimai?



Paskaitos darbas





Paskaitos darbas

Paskaitoje atliksime užduotis, tokia eiga:

1. Sprendžiame užduotis savarankiškai
2. Po savarankiško sprendimo laiko (~10 min.) dėstytojas išsprendžia 1 užduotį argumentuodamas sprendimą
3. Studentai užduoda klausimus apie sprendimą
4. Sprendimų palyginimas
5. Atliekama sekanti užduotis

Jeigu išsprendėte užduotį anksčiau nei kiti, spręskite sekančias užduotis.

Užduoties aptarimo metu, nesidrovėkite klausti kuo daugiau klausimų.
Nebūtinai jūsų sprendimas yra prastesnis. Galbūt net geresnis?

Iki kito karto!

