vaScript dasturchilari uchun TypeScript

TypeScript JavaScript bilan noodatiy munosabatda. TypeScript JavaScript-ning barcha xususiyatlarini va ularning ustiga qo'shimcha qatlamni taqdim etadi: TypeScript-ning turdagi tizimi.

Masalan, JavaScript `string` va kabi ibtidoiy tillarni taqdim etadi `number`, lekin siz ularni doimiy ravishda tayinlaganligingizni tekshirmaydi. TypeScript qiladi.

Bu sizning mavjud ishlaydigan JavaScript kodingiz ham TypeScript kodi ekanligini anglatadi. TypeScript-ning asosiy afzalligi shundaki, u sizning kodingizdagi kutilmagan xatti-harakatlarni ta'kidlab, xatolar ehtimolini kamaytiradi.

Ushbu o'quv qo'llanma TypeScript haqida qisqacha ma'lumot beradi, uning turlari tizimiga e'tibor qaratadi.

Xulosa bo'yicha turlar

TypeScript JavaScript tilini biladi va ko'p hollarda siz uchun turlarni yaratadi. Masalan, o'zgaruvchini yaratishda va uni ma'lum bir qiymatga tayinlashda, TypeScript qiymatdan uning turi sifatida foydalanadi.

```
let helloWorld = "Hello World";
         let helloWorld: string
```

JavaScript qanday ishlashini tushungan holda, TypeScript JavaScript kodini qabul qiladigan, lekin turlariga ega bo'lgan turdagi tizimni yaratishi mumkin. Bu sizning kodingizda turlarni aniq ko'rsatish uchun qo'shimcha belgilar qo'shmasdan tur tizimini taklif qiladi. TypeScript yuqoridagi misolda `helloWorld` shunday ekanligini biladi `string`.

Siz JavaScript-ni Visual Studio Code-da yozgan bo'lishingiz va muharrirni avtomatik to'ldirishga ega bo'lishingiz mumkin. Visual Studio Code JavaScript bilan ishlashni osonlashtirish uchun qopqoq ostida TypeScript-dan foydalanadi.

Turlarni aniqlash

JavaScript-da turli xil dizayn naqshlaridan foydalanishingiz mumkin. Biroq, ba'zi dizayn naqshlari turlarini avtomatik ravishda chiqarishni qiyinlashtiradi (masalan, dinamik dasturlashdan foydalanadigan naqshlar). Ushbu holatlarni qoplash uchun TypeScript JavaScript tilining kengaytmasini qo'llab-quvvatlaydi, bu sizga TypeScript-ga qanday turlar bo'lishi kerakligini aytib berish uchun joylarni taklif qiladi.

name: string Masalan, va ni o'z ichiga olgan taxmin qilingan turdagi ob'ektni yaratish uchun id: number siz quyidagilarni yozishingiz mumkin:

```
const user = {
  name: "Hayes",
  id: 0,
};
```

Deklaratsiya yordamida ushbu ob'ektning shaklini aniq tasvirlashingiz mumkin interface :

```
interface User {
  name: string;
  id: number;
}
```

Keyin JavaScript ob'ekti o'zgaruvchi deklaratsiyasidan keyingi interface sintaksisdan foydalanib, yangi ob'ektingiz shakliga mos kelishini e'lon qilishingiz mumkin : : TypeName

```
const user: User = {
  name: "Hayes",
  id: 0,
};
```

Agar siz taqdim etgan interfeysga mos kelmaydigan ob'ektni taqdim qilsangiz, TypeScript sizni ogohlantiradi:

```
interface User {
  name: string;
  id: number;
}

const user: User = {
  username: "Hayes",

Type '{ username: string; id: number; }' is not assignable to type 'User'.
  Object literal may only specify known properties, and 'username' does not
exist in type 'User'.
```

```
exist in type 'User'.
exist in type 'User'.

  id: 0,
};
```

JavaScript sinflarni va ob'ektga yo'naltirilgan dasturlashni qo'llab-quvvatlaganligi sababli, TypeScript ham shunday. Siz sinflar bilan interfeys deklaratsiyasidan foydalanishingiz mumkin:

```
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}

const user: User = new UserAccount("Murphy", 1);
```

Parametrlarni izohlash va funksiyalarga qiymatlarni qaytarish uchun interfeyslardan foydalanishingiz mumkin:

```
function getAdminUser(): User {
  //...
}

function deleteUser(user: User) {
  // ...
}
```

JavaScript-da allaqachon kichik bir ibtidoiy turlar to'plami mavjud: boolean , bigint , null , number , string , symbol , va undefined , siz ulardan interfeysda foydalanishingiz mumkin. TypeScript bu ro'yxatni yana bir nechta ro'yxat bilan kengaytiradi, masalan, any (hamma narsaga ruxsat berish), unknown(bu turni ishlatadigan kishi qaysi tur ekanligini e'lon qilishiga ishonch hosil qiling), never(bu tur sodir bo'lishi mumkin emas) va (qaytaruvchi yoki qaytarilmaydigan void funksiya) undefined qiymati).

Bino turlari uchun ikkita sintaksis mavjudligini ko'rasiz: Interfeyslar va Turlar . Siz afzal ko'rishingiz kerak interface . type Muayyan xususiyatlar kerak bo'lganda foydalaning .

Kompozitsiya turlari

TypeScript yordamida siz oddiylarni birlashtirib, murakkab turlarni yaratishingiz mumkin. Buning ikkita mashhur usuli bor: kasaba uyushmalari va generiklar bilan.

Uyushmalar

Birlashma bilan siz tur ko'p turlardan biri bo'lishi mumkinligini e'lon qilishingiz mumkin. Misol uchun, siz turni yoki boolean quyidagicha tasvirlashingiz mumkin : true false

```
type MyBool = true | false;
```

Eslatma: Agar kursorni yuqoriga olib kelsangiz MyBool , u sifatida tasniflanganligini ko'rasiz boolean . Bu Strukturaviy turdagi tizimning xususiyati. Bu haqda quyida batafsilroq.

Birlashma turlaridan foydalanishning mashhur usuli bu qiymatga ruxsat berilgan to'plam string yoki number harflarni tavsiflashdir :

```
type WindowStates = "open" | "closed" | "minimized";
type LockStates = "locked" | "unlocked";
type PositiveOddNumbersUnderTen = 1 | 3 | 5 | 7 | 9;
```

Kasaba uyushmalari har xil turlarni ham hal qilish yo'lini taqdim etadi. Masalan, sizda array yoki a oladigan funksiya bo'lishi mumkin string :

```
function getLength(obj: string | string[]) {
  return obj.length;
}
```

O'zgaruvchining turini o'rganish uchun quyidagilardan foydalaning typeof :

Turi	Predikat
ip	`typeof s === "string"`

Turi	Predikat
raqam	`typeof n === "number"`
mantiqiy	`typeof b === "boolean"`
aniqlanmagan	`typeof undefined === "undefined"`
funktsiyasi	`typeof f === "function"`
massiv	`Array.isArray(a)`

Masalan, siz funktsiyaga satr yoki massiv uzatilganligiga qarab turli qiymatlarni qaytarishingiz mumkin:

```
function wrapInArray(obj: string | string[]) {
  if (typeof obj === "string") {
    return [obj];
             (parameter) obj: string
  }
  return obj;
}
```

Umumiy

Generiklar turlarga o'zgaruvchilarni beradi. Umumiy misol - massiv. Jeneriksiz massiv har qanday narsani o'z ichiga olishi mumkin. Umumiy ma'lumotlarga ega massiv massiv o'z ichiga olgan qiymatlarni tavsiflashi mumkin.

```
type StringArray = Array<string>;
type NumberArray = Array<number>;
type ObjectWithNameArray = Array<{ name: string }>;
```

Siz generiklardan foydalanadigan o'z turlaringizni e'lon qilishingiz mumkin:

```
interface Backpack<Type> {
  add: (obj: Type) => void;
  get: () => Type;
}
```

```
// This line is a shortcut to tell TypeScript there is a
// constant called `backpack`, and to not worry about where it came from.
declare const backpack: Backpack<string>;

// object is a string, because we declared it above as the variable part of Bac
const object = backpack.get();

// Since the backpack variable is a string, you can't pass a number to the add
backpack.add(23);
```

```
Argument of type 'number' is not assignable to parameter of type 'string'.
```

Strukturaviy tip tizimi

TypeScript-ning asosiy tamoyillaridan biri shundan iboratki, tipni tekshirish qadriyatlarga ega bo'lgan *shaklga* qaratiladi . Bu ba'zan "o'rdak yozish" yoki "strukturali yozish" deb ataladi.

Strukturaviy tipdagi tizimda ikkita ob'ekt bir xil shaklga ega bo'lsa, ular bir xil turdagi hisoblanadi.

```
interface Point {
  x: number;
  y: number;
}

function logPoint(p: Point) {
  console.log(`${p.x}, ${p.y}`);
}

// logs "12, 26"
const point = { x: 12, y: 26 };
logPoint(point);
```

O'zgaruvchi `point` hech qachon tip deb e'lon qilinmaydi `Point` . Biroq, TypeScript turini tekshirishda ning `point` shaklini shakli bilan solishtiradi . `Point` Ular bir xil shaklga ega, shuning uchun kod o'tadi.

Shaklni moslashtirish faqat mos keladigan ob'ekt maydonlarining kichik to'plamini talab qiladi.

```
const point3 = { x: 12, y: 26, z: 89 };
logPoint(point3); // logs "12, 26"

const rect = { x: 33, y: 3, width: 30, height: 80 };
logPoint(rect); // logs "33, 3"
```

```
const color = { hex: "#187ABF" };
logPoint(color);

Argument of type '{ hex: string; }' is not assignable to parameter of type
'Point'.
  Type '{ hex: string; }' is missing the following properties from type
'Point': x, y
```

Sinflar va ob'ektlar shakllarga qanday mos kelishi o'rtasida farq yo'q:

```
class VirtualPoint {
  x: number;
  y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }
}

const newVPoint = new VirtualPoint(13, 56);
logPoint(newVPoint); // logs "13, 56"
```