Lekcja 9: any, wszystko w porządku?

W Lekcji 8 utworzyliśmy pierwsze adnotacje typu w funkcjach. Adnotacje typu mogą być dowolnego typu pierwotnego, a także typów kompozycji, które istnieją w JavaScript. I jest więcej, które są wyłączne dla TypeScript. Jak any.

any jest typem domyślnym, jeśli nie określimy typu ani nie pozwolimy, aby TypeScript go wywnioskował.

let deliveryAddress // deliveryAddress jest any

W momencie przypisania wartości typ staje się bardziej szczegółowy:

// deliveryAddress jest typu string

let deliveryAddress = '421 Smashing Hill, 90210'

Chyba że wyraźnie określimy typ za pomocą adnotacji:

// deliveryAddress jest typu any

let deliveryAddress: any = '421 Smashing Hill, 90210'

Oczywiście działa to tylko wtedy, gdy typy są zgodne. aby może być wszystkim: to topowy typ i dlatego wszystkie inne typy są częścią any. Jeśli przypiszemy niekompatybilną wartość do zmiennej określonego typu, TypeScript rzuci nam faliste linie:

// deliveryAddress jest typu string,

// po co przypisywać numer?

let deliveryAddress: string = 2;

Pisanie od lewej a pisanie od prawej

Jawna adnotacja typu zawsze jest pierwsza. W momencie, gdy adnotujemy za pomocą składni dwukropka, nazwa jest typu, który opisujemy. Wszystkie wartości muszą być zgodne i muszą być zgodne. Nazywamy tę technikę pisaniem lewą ręką, ponieważ pisanie odbywa się przed (po lewej) znaku równości. Pomijanie adnotacji typu i praca w pierwszej kolejności z wnioskami o typie nazywa się pisaniem prawostronnym: wpisywanie odbywa się po prawej stronie znaku równości, czy to poprzez przypisanie wartości i wnioskowanie, czy też przez typ wartości zwracanej przez funkcję. To samo dotyczy typów kompozycji:

let deliveryAddresses = [

„421 Smashing Hill, 90210”,

„221b Paw-ker Street”,

"4347 Whiskers-ia Lane",

]

// deliveryAddresses to ciąg []

Jest to tablica ciągów, więc typ to string []. Możemy jawnie wpisać deliveryAddresses do string [].

let deliveryAddresses: string [] = []

// DOBRZE

deliveryAddresses.push ('421 Smashing Hill, 90210')

// Nie w porządku! 2 nie jest ciągiem

deliveryAddresses.push (2000)

Pisanie zarówno od lewej , jak i od prawej ma swoje wady i zalety. Dzięki pisaniu lewą ręką możesz dużo pomyśleć o swoich typach, zanim zaczniesz kodować resztę. Pisanie prawą ręką umożliwia tworzenie typów na bieżąco, co może być nieco bardziej JavaScript-y.

Problem z any

Mimo że any jest podstawą dla wszystkich typów w TypeScript i jest domyślnym dla wszystkiego, z czego TypeScript nie może wywnioskować typów, rzadko będziesz musiał zadeklarować coś jako any. Zwykle chcesz mieć więcej informacji o swoich typach, a nie mniej. W każdym przypadku takie rzeczy są możliwe:

const myName: any = 'Kot Fritz'

myName.firstLetter.makeCapitals ()

Oczywiście właściwości takie jak firstLetter i funkcje takie jak makeCapitals nie istnieją w zwykłych typach JavaScript. Ale każdy o tym nie wie i nie przejmuje się tym. W każdym przypadku wszystkie umowy są spełnione:

1. Typ myName jest any, dzięki adnotacji po lewej stronie. „Kot Fritz" jest ciągiem znaków, ale można go przypisać do any. Więc to zadanie jest OK!

2. Ponieważ aby może być dowolne, any umożliwia nam również dostęp do właściwości, których może nie być. Mogą tam być - w końcu może to być wszystko! To oczywiście kompletny nonsens, ale to właśnie otrzymujesz, pracując z any.

any to symbol wieloznaczny! Użyj any, a możesz iść na całość i w ogóle zapomnieć o sprawdzaniu typów. Dlaczego więc coś takiego jak any w ogóle istnieje? Wynika to z natury JavaScript. W JavaScript nie jesteś powiązany z typem, a wartości dowolnego typu mogą pojawiać się w zmiennych i właściwościach. Niektórzy programiści nadmiernie to wykorzystują! any odzwierciedla nadrzędną elastyczność JavaScript; możesz postrzegać to jako furtkę do świata, w którym nie potrzebujesz narzędzi ani bezpieczeństwa typu. Jak najbardziej używaj any, ale zrozum, jak to działa i co z nim zrobić - używaj go na własne ryzyko!

Upewnij się, że chcesz użyć any jawnie jako adnotacji typu. A jeśli chcesz wejść przez tylne drzwi do elastyczności JavaScript, bądź bardzo celowy poprzez rzutowanie typów:

// theObject to obiekt, dla którego nie mamy typu,

// ale wiemy dokładnie co

// my robimy!

(theObject jak dowolny) .firstLetter.toUppercase ()

Oczywiście odlewy typów działają również z innymi typami. Jeśli chcesz się upewnić, że nie masz w swoim kodzie niczego, czego się nie spodziewasz, ustaw flagę kompilatora noImplicitAny na true.

TypeScript następnie upewni się, że przypisujesz wartości do prawidłowego wnioskowania typów lub, w przypadku jakichkolwiek, upewni się, że jawnie opatrzysz adnotacjami lub rzutujesz na dowolny.

```
// deliveryAddress jest typu any, ponieważ my

// nie opisał konkretnego typu. Wątpliwe są

// Trudno później wyśledzić, dlatego to dobrze

// żeby krzyczeć na nas TypeScript

function printAddress (deliveryAddress) {

console.log (deliveryAddress)

}
```

Jeśli oznaczymy parametry funkcji i zmienne jawnie any, łatwiej będzie je później wyśledzić, gdy już zdecydujemy się na typy rzeczywiste.