Lekcja 3: Typy

Znaleźliśmy już kilka błędów w naszej ostatniej lekcji, ale jeszcze nie skończyliśmy. Istnieje wiele innych problemów do znalezienia! Aby dostać się do dna beczki, musimy być trochę bardziej przemyślani co do naszych typów. Poczekaj sekundę.

Co to w ogóle są typy?

W swojej książce Programming with Types Vlad Riscutia definiuje typy w następujący sposób:

Typ to klasyfikacja danych, która definiuje operacje, które można wykonać na tych danych, znaczenie danych i zbiór dozwolonych wartości. Wpisywanie jest sprawdzane przez kompilator i / lub w czasie wykonywania, aby zapewnić integralność danych, wymusić ograniczenia dostępu i zinterpretować dane zgodnie z zamysłem programisty.

W przypadku typów wiemy, że „Hello World" to ciąg znaków, 1234 to typ liczbowy, a true to Boolean. Bez typów typu boolean, string lub number te wartości byłyby po prostu zerami i jedynkami w pamięci jakiegoś komputera i nie wiedzielibyśmy, co z nimi zrobić. Typ nie tylko określa, jak należy interpretować wartość, ale także nagle dostarcza nam operacji! Możemy pomnożyć 1,234 przez 10 lub uczynić „Hello World" wielkimi literami za pomocą .toUpperCase (). Typy są podstawą programowania. JavaScript ma typy! I nie pozwól nikomu powiedzieć Ci inaczej! W JavaScript masz typy prymitywne, takie jak number, string i boolean. Tak, są inne typy prymitywne i dojdziemy do nich z czasem. Istnieją również złożone typy danych, takie jak:

• obiekty: właściwości i wartości różnych typów pierwotnych i złożonych

• tablice: listy wartości, które mogą mieć dowolny typ

• funkcje: metody, które wywołujesz z parametrami pewnych typów i które zwracają wartości określonych typów

Są symbole, ale to zupełnie inna historia.

Słabo napisane

JavaScript jest jednak napisany słabo. Oznacza to, że nawet jeśli utworzysz zmienną lub właściwość i przypiszesz wartość określonego typu, możesz przełączać typy w trakcie:

niech val = 1234; // DOBRZE!

val = 'Onetwothreefour'; // Zmiana przypisania. Nadal ok!

Możesz także bez problemu łączyć wartości różnych typów:

niech val = {a: 3} + 5

// [obiekt Object] 5! Co to w ogóle znaczy?

Wyniki niekoniecznie muszą mieć sens. I chociaż takie operacje są możliwe, powszechnie uważa się je za błędy - błędy, których nie można wykryć.

Silnie napisane

TypeScript jest silnie wpisany. Oznacza to, że po przypisaniu wartości określonego typu TypeScript chce, abyś się z nią go trzymał.

niech val = 1234; // DOBRZE! val to liczba

// Czekaj, co? Teraz to jest ciąg? To niemożliwe!

val = 'Onetwothreefour';

Oznacza to również, że nie możesz używać operatorów na wartościach różnych typów, ponieważ zazwyczaj nie mają one sensu:

// Chcesz dodać liczbę do obiektu? Dlaczego??

let val = { a: 3 } + 5

A dzięki temu otrzymujesz wszystkie czerwone faliste linie. Jeśli jesteś przyzwyczajony do ponownego używania zmiennych lub dodawania liczb i ciągów w locie, możesz czuć się nieco ograniczony podczas korzystania z TypeScript. Wymieniasz tę elastyczność na kod, który jest o wiele bardziej dźwiękowy, poprawny i pozbawiony potencjalnych pułapek.

Kształty

Typy prymitywne są raczej łatwe do zdefiniowania. Mogą mieć bardzo określony zakres wartości. Na przykład typ logiczny jest najprostszy: może mieć wartość true, false, undefined lub null. Możesz policzyć liczbę możliwych wartości palcami. Liczby i ciągi znaków pozwalają na znacznie więcej wartości, ale tylko na taką ilość pamięci i bieżące środowisko wykonawcze JavaScript pozwalają (sprawdź Number.MIN_VALUE i Number.MAX_VALUE, aby poznać numer zakresu). To staje się bardziej złożone w przypadku typów złożonych. Tutaj mamy do czynienia nie tylko z zakresami wartości, ale także z tzw. Kształtami. Kształty mówią nam więcej o cechach strukturalnych typu: typy i nazwy właściwości obiektu, typy i nazwy parametrów funkcji, typy i indeksy elementów tablicy. Weźmy na przykład następujący obiekt osoby:

const person = {

firstName: 'Stefan',

lastName: 'Baumgartner',

age: 38

}

person jest typu object, ale podąża za kształtem: firstName jest łańcuchem; lastName to ciąg; a age to liczba. Możemy zdefiniować ten kształt jako typ niestandardowy:

type Person = {

firstName: string,

lastName: string,

age: number

}

Ponieważ TypeScript jest strukturalnym systemem typów, kształty są niezwykle ważne. Dopóki zmienne pasują do określonego kształtu (w taki sposób, że {writable: false} pasuje do kształtu PropertyDescriptor w naszym poprzednim przykładzie), TypeScript będzie działał poprawnie z twoim kodem. Jeśli nadasz obiektom różne kształty, pojawi się jeszcze więcej błędów. Sprawdźmy to na następnej lekcji.