
TypeScript

Skuteczne programowanie

62 sposoby ulepszania kodu TypeScript

Dan Vanderkam

przekład: Joanna Zatorska

Spis treści

Wstęp	ix
1. Poznajemy TypeScript.....	1
Element 1: Relacja między TypeScript a JavaScript.....	1
Element 2: Które opcje TypeScript wykorzystujemy.....	7
Element 3: Generowanie kodu jest niezależne od typów.....	10
Element 4: Przyzwyczaj się do strukturalnego typowania.....	17
Element 5: Ograniczanie użycia typu any.....	22
2. System typowania TypeScript	27
Element 6: Używanie edytora do sprawdzania i eksploracji systemu typowania	27
Element 7: Typy jako zbiory wartości.....	31
Element 8: Sprawdzanie czy symbol należy do przestrzeni typu czy do przestrzeni wartości.....	38
Element 9: Deklarujemy typy zamiast stosować asercje typów.....	43
Element 10: Unikajmy typów opakowujących obiekty (String, Number, Boolean, Symbol, BigInt).....	47
Element 11: Limity testowania dodatkowych właściwości.....	50
Element 12: Stosujemy typy w całych wyrażeniach funkcyjnych, gdy jest to możliwe.....	53
Element 13: Odróżniamy typ od interfejsu.....	56
Element 14: Używajmy operacji typów i typów generycznych, aby uniknąć powtórzeń.....	61
Element 15: Korzystajmy z sygnatur indeksów dla danych dynamicznych.....	69
Element 16: Używajmy tablic, krotek i typu ArrayLike zamiast liczbowych sygnatur indeksów.....	73
Element 17: Używajmy readonly, aby uniknąć błędów związanych z mutowaniem	77
Element 18: Korzystajmy z typów mapowanych, aby zapewnić synchronizację wartości.....	83

3. Wnioskowanie typów	87
Element 19: Unikajmy zaśmiecania kodu typami, które można wywnioskować ...	87
Element 20: Używajmy różnych zmiennych dla różnych typów	94
Element 21: Rozszerzanie typów	97
Element 22: Zawężanie typów	100
Element 23: Tworzenie całych obiektów od razu.	104
Element 24: Zachowajmy spójność w używaniu aliasów	107
Element 25: Używajmy funkcji asynchronicznych zamiast asynchronicznych funkcji zwrotnych	110
Element 26: Jak wykorzystuje się kontekst podczas wnioskowania typów	116
Element 27: Korzystajmy z konstrukcji funkcyjnych i bibliotek, aby ułatwić przepływ typów	120
4. Projektowanie typów	125
Element 28: Preferujmy typy, które zawsze reprezentują poprawne stany	125
Element 29: Liberalne podejście do akceptowanych wartości i surowe do zwracanych danych	131
Element 30: Nie powtarzajmy informacji o typie w dokumentacji.	134
Element 31: Przenośmy wartości null poza obręb swojego typu.	136
Element 32: Używajmy unii interfejsów zamiast interfejsów unii.	140
Element 33: Preferujmy bardziej precyzyjne alternatywy typów łańcuchowych ..	144
Element 34: Używajmy niekompletnych typów zamiast nieprecyzyjnych typów. .	148
Element 35: Generujmy typy na podstawie API i specyfikacji a nie danych.	153
Element 36: Nazywajmy typy zgodnie z językiem domeny swojego projektu.	158
Element 37: Rozważmy stosowanie „etykiet” dla typowania nominalnego.	161
5. Korzystanie z typu any	165
Element 38: Używanie możliwie największego zakresu dla typów any	165
Element 39: Preferujmy bardziej precyzyjne warianty od zwykłego typu any ...	168
Element 40: Ukrywajmy nie gwarantujące bezpieczeństwa asercje typów w typowanych funkcjach	170
Element 41: Ewolucja typu any	172
Element 42: Korzystajmy z typu unknown zamiast z typu any w przypadku wartości nieznanego typu.	175
Element 43: Stosujmy mechanizmy bezpieczne pod kątem typów zamiast metody monkey patching.	179
Element 44: Śledźmy pokrycie typami, aby zapobiec regresji w bezpieczeństwie typów	182

6. Deklaracje typów i składnia @types	185
Element 45: Umieścimy TypeScript i @types w deklaracjach devDependencies. . .	185
Element 46: Trzy numery wersji w deklaracjach typów	187
Element 47: Eksportujemy wszystkie typy z publicznych interfejsów API	191
Element 48: Używajmy TSDoc do tworzenia komentarzy API	192
Element 49: Zdefiniujemy typ dla elementu this w funkcjach zwrotnych	195
Element 50: Używajmy typów warunkowych zamiast przeciążonych deklaracji . .	200
Element 51: Odzwierciedlajmy typy dla zależności.	202
Element 52: Pamiętajmy o pułapkach testowania typów	204
7. Pisanie i uruchamianie kodu	209
Element 53: Korzystajmy z funkcjonalności standardu ECMAScript zamiast TypeScript.	209
Element 54: Iterowanie obiektów	214
Element 55: Hierarchia DOM	217
Element 56: Nie opierajmy się na zmiennych prywatnych w celu ukrycia informacji	222
Element 57: Korzystajmy z map kodu źródłowego do debugowania kodu TypeScript.	225
8. Migracja do TypeScript	231
Element 58: Tworzenie nowoczesnego kodu JavaScript	232
Element 59: Używajmy składni @ts-check oraz dokumentacji JSDoc w eksperymentach z TypeScript	240
Element 60: Używajmy allowJs, aby połączyć kod TypeScript z JavaScript	245
Element 61: Przekształcajmy moduły po kolei wędrując w górę grafu zależności .	246
Element 62: Nie uważajmy migracji za zakończoną przed włączeniem opcji noImplicitAny	251
Indeks	255