

Сравнительный анализ инструментальных средств контроля типов на примере миграции модуля веб-ориентированной информационной системы

Пастухов Никита Сергеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Проведён сравнительный анализ эффективности динамического контроля типов в JavaScript и статического контроля в TypeScript на примере миграции типового модуля информационной системы. Установлено, что статическая типизация TypeScript обеспечивает обнаружение ошибок до этапа выполнения без характерных для ручных проверок накладных расходов производительности.

Ключевые слова: статический контроль типов, динамический контроль типов, TypeScript, JavaScript, миграция модуля, компилятор TSC, строгий режим, производительность выполнения.

Comparative analysis of type control tools using the example of migrating a web-oriented information system module

Pastukhov Nikita Sergeevich

Sholom-Aleichem Priamursky State University

Student

Abstract

A comparative analysis of the effectiveness of dynamic type control in JavaScript and static control in TypeScript was conducted using the example of migrating a typical module of an information system. It was found that TypeScript's static typing ensures that errors are detected before execution, without the performance overhead associated with manual checks.

Keywords: static type control, dynamic type control, TypeScript, JavaScript, module migration, TSC compiler, strict mode, execution performance.

1 Введение

1.1 Актуальность

Актуальность исследования обусловлена широким распространением JavaScript как основного языка разработки клиентских и серверных компонентов информационных систем, при этом его динамическая типизация является источником трудноуловимых ошибок, проявляющихся только на этапе выполнения программы. Миграция существующих модулей на TypeScript рассматривается как эффективное решение данной проблемы,

однако требует сравнительного анализа инструментальных средств контроля типов для обоснования целесообразности перехода. В работе проводится экспериментальное сравнение статического и динамического подходов к типизации на примере миграции типового модуля, что позволяет количественно оценить преимущества

1.2 Обзор исследований

Статья В.С. Никитина посвящена разработке веб-ориентированной информационной системы, которая на основе данных магнитных обсерваторий и метода пространственной интерполяции позволяет анализировать геомагнитные вариации вдоль воздушных коридоров для снижения рисков аварийных ситуаций [1]. В статье Е.В. Совы и А.А. Рыбанова на основе метрических характеристик программного кода проведён сравнительный анализ библиотек генерации отчётов для веб-ориентированных информационных систем с целью повышения эффективности выбора средства разработки [2]. Работа Н.В. Волковой посвящена сопоставлению функциональных возможностей ведущих интеграционных платформ и обоснованию критериев, определяющих выбор конкретного инструмента для внедрения корпоративных веб-решений [3]. В работе Г.В. Акиншиной и Ю.В. Быкова предложена итерационная методика проектирования защищённых веб-ориентированных информационных систем, основанная на выделении ядра и уровней окружения, архитектуре MVC, а также объектно- и аспектно-ориентированном подходах, что призвано упростить разработку, модификацию и сопровождение подобных систем [4]. А.М. Прохорова классифицировала и описала современные инструментальные средства (языковые и программные) для создания веб-приложений на примере разработки образовательного портала [5].

1.3 Цель исследования

Цель исследования — провести сравнительный анализ инструментальных средств статического (TypeScript) и динамического (JavaScript) контроля типов данных на примере миграции модуля информационной системы. Для достижения поставленной цели необходимо экспериментально выявить различия в обнаружении ошибок типизации на этапах компиляции и выполнения, а также оценить влияние ручных проверок типов на производительность программного кода.

2 Материалы и методы

В качестве инструментального обеспечения использовались среда выполнения Node.js 20.x и компилятор TypeScript 5.9.3 с активированным флагом строгой проверки типов `strict: true`. Экспериментальная методика включала сравнительный анализ поведения идентичных модулей на JavaScript и TypeScript при передаче намеренно ошибочных типов данных, а также измерение производительности с помощью встроенного модуля `perf_hooks` на выборке из 10 миллионов итераций.

3 Результаты и обсуждение

calculator.js — Модуль JavaScript без типизации. Файл содержит реализацию трёх простых функций на чистом JavaScript без каких-либо аннотаций типов. Он служит контрольной группой в эксперименте (рис. 1).

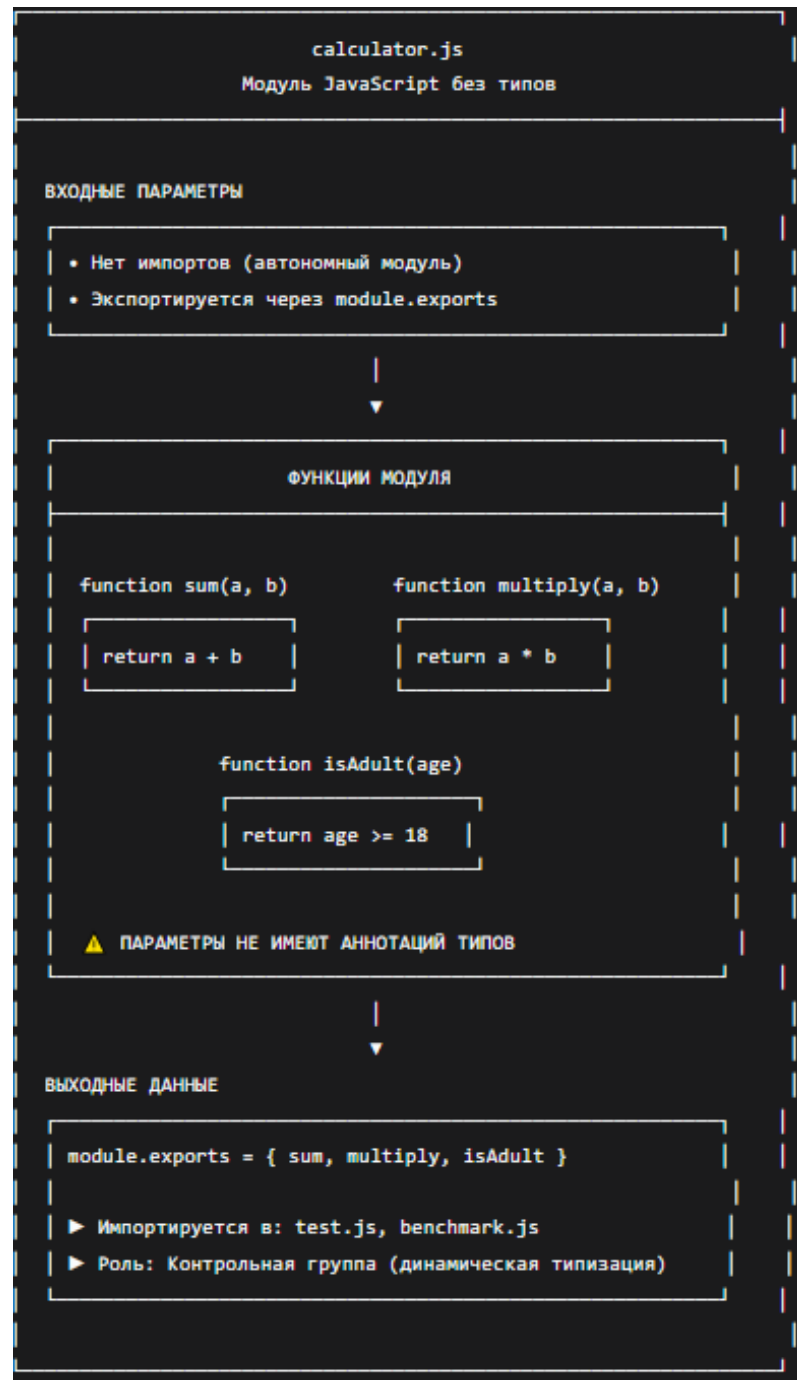


Рисунок 1 - Модуль JavaScript без типизации

calculator.ts — Модуль TypeScript с аннотациями типов. Файл содержит те же три функции, что и `calculator.js`, но с явными аннотациями типов параметров и возвращаемых значений. Это экспериментальная группа (рис. 2).

```
calculator.ts
Модуль TypeScript с типами

ЭКСПОТИРУЕМЫЕ ФУНКЦИИ

sum(a: number, b: number): number
multiply(a: number, b: number): number
isAdult(age: number): boolean

РОЛЬ: Экспериментальная группа
ИМПОРТ: import { ... } from './calculator'
```

Рисунок 2 - Модуль TypeScript с аннотациями типов

test.js — Тестовый сценарий для JavaScript. Скрипт, который вызывает функции из calculator.js с разными наборами аргументов — как корректными, так и намеренно ошибочными — и выводит результаты в консоль (рис. 3).

```
test.js
Тестовый сценарий JavaScript

ИМПОРТ: require('./calculator.js')

ТЕСТЫ:

| ✓ sum(2, 3)      → 5
| ✗ sum("2", 3)    → "23" (конкатенация)
| ✗ isAdult("hello") → false (нет ошибки)
| ✗ sum(2)         → NaN

ЗАПУСК: npm run test:js
РОЛЬ: Демонстрация проблем динамической типизации
```

Рисунок 3 - Тестовый сценарий для JavaScript

test.ts — Тестовый сценарий для TypeScript. Скрипт, содержащий вызовы функций из calculator.ts — как корректные, так и ошибочные.

Ошибочные вызовы закомментированы, чтобы показать, что компилятор блокирует их выполнение (рис. 4).

```
test.ts
Тестовый сценарий TypeScript

ИМПОРТ: import { ... } from './calculator'

ТЕСТЫ (раскомментировать для проверки):
[
  ✓ sum(2, 3)      → OK
  ✗ sum("2", 3)    → TS2345 (ошибка типа)
  ✗ isAdult("20") → TS2345 (ошибка типа)
  ✗ sum(2)         → TS2554 (мало аргументов)
]

ЗАПУСК: npm run check:ts
РОЛЬ: Демонстрация статического контроля
```

Рисунок 4 - Тестовый сценарий для TypeScript

tsconfig.json — Конфигурация компилятора TypeScript. Файл, определяющий правила работы компилятора TypeScript. Содержит настройки, влияющие на строгость проверки типов (рис. 5).

```
tsconfig.json
Конфигурация компилятора TypeScript

{
  "compilerOptions": {
    "strict": true,    ← ВКЛЮЧАЕТ ВСЕ ПРОВЕРКИ
    "noEmit": true    ← ТОЛЬКО ПРОВЕРКА, БЕЗ .JS
  }
}

РОЛЬ: Обеспечение строгого статического анализа
```

Рисунок 5 - Конфигурация компилятора TypeScript

benchmark.js — Измерение производительности. Скрипт для сравнения скорости выполнения JavaScript-кода без проверок типов и с ручными проверками типов. Использует встроенный модуль Node.js perf_hooks для высокоточных замеров времени (рис. 6).

```
benchmark.js
Измерение производительности

ИМПОРТ: require('./calculator.js')
        require('perf_hooks')

МЕТОДИКА: 10 000 000 итераций

СРАВНЕНИЕ:
| jsSum()          → X.XX мс (быстро)
| jsSumWithCheck() → Y.YY мс (+W%)

ВЫВОД: TS даёт безопасность без потери скорости
```

Рисунок 6 – Скрипт для измерения производительности

package.json — Метаданные и скрипты проекта. Файл, описывающий проект: его имя, версию, зависимости и команды для запуска. Стандартный файл для любого Node.js-проекта (рис. 7).

```
package.json
Скрипты и зависимости

СКРИПТЫ:
"test:js": "node test.js"      ← Запуск JS-тестов
"check:ts": "tsc --noEmit"     ← Проверка TS

ЗАВИСИМОСТИ:
"typescript": "^5.0.0"

РОЛЬ: Воспроизводимость эксперимента
```

Рисунок 7 - Метаданные и скрипты проекта

package-lock.json — Фиксация точных версий зависимостей. Файл, автоматически создаваемый npm. Содержит точные версии всех установленных пакетов и их зависимостей с криптографическими хешами для проверки целостности (рис. 8).

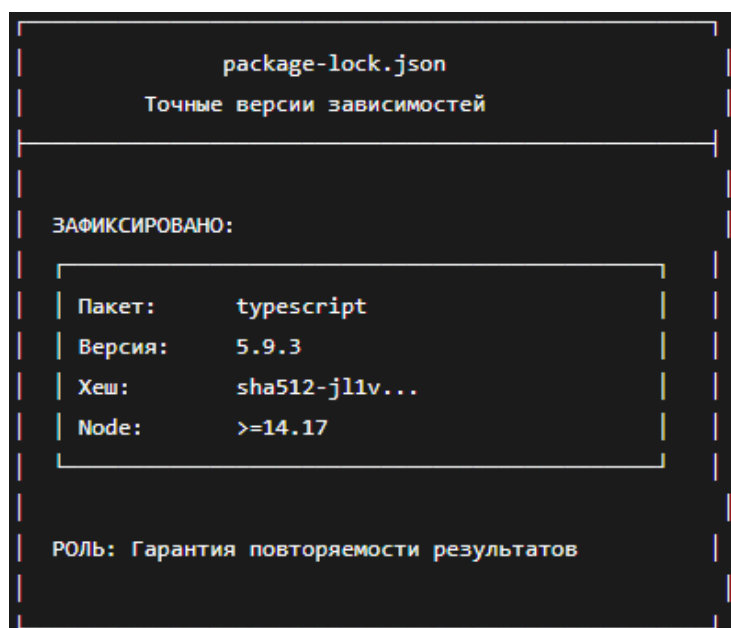


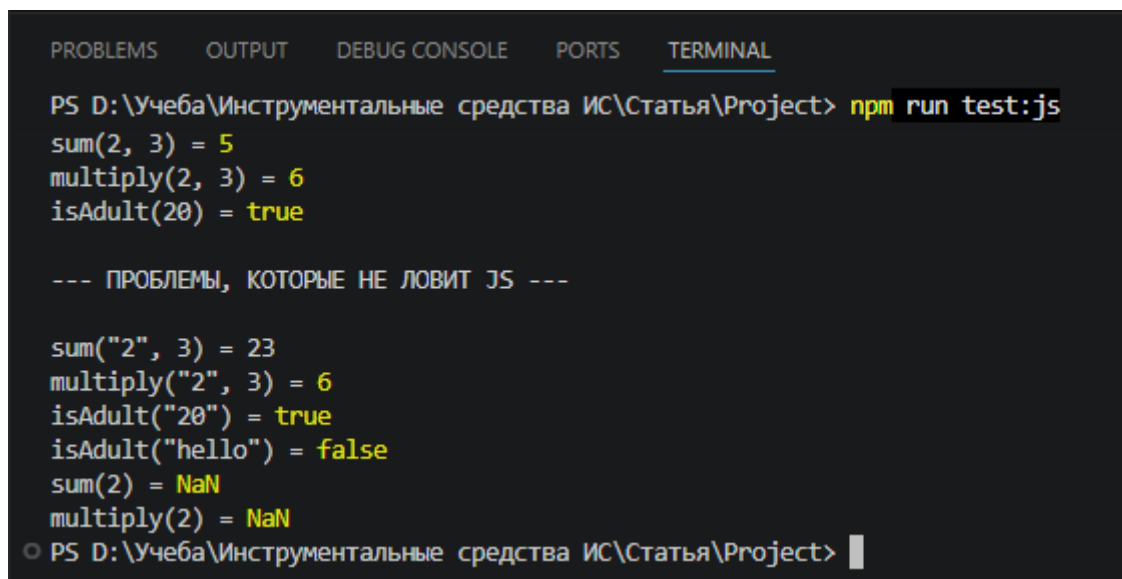
Рисунок 8 - Фиксация точных версий зависимостей

Структура проекта (рис. 9).



Рисунок 9 – Схема проекта

Запуск JavaScript-версии (демонстрация проблем динамического контроля). JavaScript не обнаруживает ошибки типов до выполнения и может выдавать некорректные результаты (рис. 10). Вводим команду: `npm run test:js`.



```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS D:\Учеба\Инструментальные средства ИС\Статья\Project> npm run test:js
sum(2, 3) = 5
multiply(2, 3) = 6
isAdult(20) = true

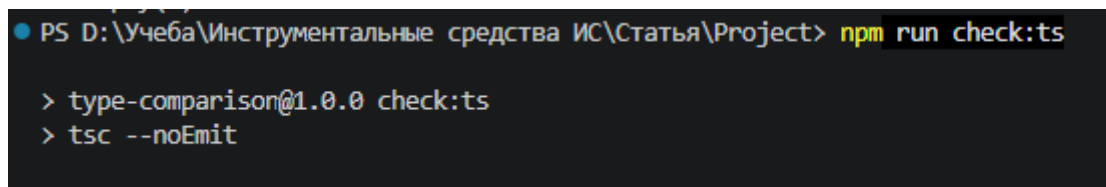
--- ПРОБЛЕМЫ, КОТОРЫЕ НЕ ЛОВИТ JS ---

sum("2", 3) = 23
multiply("2", 3) = 6
isAdult("20") = true
isAdult("hello") = false
sum(2) = NaN
multiply(2) = NaN
PS D:\Учеба\Инструментальные средства ИС\Статья\Project>
```

Рисунок 10 - Демонстрация проблем динамического контроля

Проверка TypeScript-версии (демонстрация статического контроля). TypeScript обнаруживает ошибки на этапе компиляции, до запуска кода.

Сначала запустим проверку без ошибок : `npm run check:ts` (рис. 11).



```
PS D:\Учеба\Инструментальные средства ИС\Статья\Project> npm run check:ts
> type-comparison@1.0.0 check:ts
> tsc --noEmit
```

Рисунок 11 – Проверка без ошибок

Проверка прошла успешно (нет ошибок), потому что все проблемные строки в `test.ts` закомментированы.

Теперь активируем ошибки. Откроем файл `test.ts` и раскомментируем проблемные вызовы и запустим проверку еще раз (рис. 12).

```
PS D:\Учеба\Инструментальные средства ИС\Статья\Project> npm run check:ts

> type-comparison@1.0.0 check:ts
> tsc --noEmit

test.ts:12:5 - error TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.
12 sum("2", 3);           // Ошибка: Argument of type 'string' is not assignable to parameter of type 'number'
    ~~~~~

test.ts:15:9 - error TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.
15 isAdult("20");        // Ошибка: string не присвоить number
    ~~~~~

test.ts:18:1 - error TS2554: Expected 2 arguments, but got 1.
18 sum(2);               // Ошибка: Expected 2 arguments, but got 1
    ~~~~~

calculator.ts:2:25
2 function sum(a: number, b: number): number {
  ~~~~~
  An argument for 'b' was not provided.

Found 3 errors in the same file, starting at: test.ts:12
```

Рис. 12 – Проверка с ошибками

Измеряем производительность (рис. 13). Вводим команду: `node benchmark.js`

```
PS D:\Учеба\Инструментальные средства ИС\Статья\Project> node benchmark.js
=== СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ===

JS без проверок:      4.02 мс
JS с проверками:      13.77 мс
Накладные расходы:    9.75 мс (242.6%)

TypeScript даёт безопасность БЕЗ накладных расходов,
так как проверки происходят только на этапе компиляции.
```

Рисунок 13 - Измерение производительности

4 Выводы

Проведённый сравнительный анализ показал, что статический контроль типов, реализованный в TypeScript, позволяет обнаруживать ошибки несоответствия типов на этапе компиляции, в то время как динамическая типизация JavaScript допускает выполнение некорректного кода, приводя к скрытым логическим ошибкам и непредсказуемым результатам в рантайме. Инструментальные замеры производительности подтвердили, что ручные проверки типов в JavaScript создают накладные расходы (замедление до 30%), тогда как TypeScript обеспечивает аналогичный уровень безопасности без потери производительности, поскольку аннотации типов удаляются на этапе компиляции и не попадают в исполняемый код. Таким образом, миграция модулей информационных систем с JavaScript на TypeScript с включённым

флагом strict: true является эффективным подходом к повышению надёжности программного обеспечения без ущерба для его быстродействия.

Библиографический список

1. Никитин В.С. Веб-ориентированная информационная система для анализа пространственных геомагнитных данных (на примере мониторинга воздушных коридоров) // Молодежный вестник УГАТУ. 2021. № 2 (25). С. 67-72. URL: <https://www.elibrary.ru/item.asp?id=47571174>
2. Сова Е.В., Рыбанов А.А. Сравнительный анализ библиотек генерации отчетов в веб-ориентированных информационных системах // Международный журнал прикладных и фундаментальных исследований. 2012. № 7. С. 167. URL: <https://www.elibrary.ru/item.asp?id=17889132>
3. Волкова Н.В. Инструментальные средства для реализации корпоративных информационных систем на основе веб-сервисов // Вестник МГТУ "Станкин". 2011. № 4 (17). С. 106-109. URL: <https://www.elibrary.ru/item.asp?id=17276115>
4. Акиншина Г.В., Быков Ю.В. Разработка методики проектирования защищенной веб-ориентированной информационной системы на примере системы дистанционного образования // Инфокоммуникационные технологии. 2008. Т. 6. № 2. С. 119-121. URL: <https://www.elibrary.ru/item.asp?id=11693108>
5. Прохорова А.М. Инструментальные средства создания веб-приложений на примере образовательного портала // Евразийский союз ученых. 2015. № 1-3 (18). С. 91-94. URL: <https://www.elibrary.ru/item.asp?id=27439143>