**Type Guard** — это функция или выражение, которое проверяет тип переменной и позволяет TypeScript уточнить его тип в конкретном блоке кода. Это помогает избежать ошибок, связанных с несоответствием типов, и улучшает читаемость и поддержку кода.

Тайпгарды (Type Guards) в TypeScript используются для уточнения типов переменных в условиях, когда тип переменной может быть одним из нескольких. Это позволяет компилятору TypeScript более точно определять типы и предотвращать ошибки во время выполнения. Вот несколько случаев, когда необходимо использовать тайпгарды:

1. Обработка объединенных типов: Когда переменная может быть нескольких типов (например, `string | number`), тайпгард позволяет определить конкретный тип в определенном контексте.

2. Проверка на `null` или `undefined`: Тайпгарды помогают избежать ошибок, связанных с доступом к свойствам объектов, которые могут быть `null` или `undefined`.

3. Работа с интерфейсами и классами: Когда есть несколько интерфейсов или классов с похожими свойствами, тайпгарды могут использоваться для проверки принадлежности объекта к определенному интерфейсу или классу.

4. Улучшение автодополнения и проверки типов: Использование тайпгардов улучшает автодополнение и проверку типов в IDE, делая код более безопасным и понятным.

**Встроенные тайпгарды**

**- typeof: Проверяет тип примитивных значений.**

Ограничитель типа typeof используется для определения типа переменной. Считается, что защита типа typeof является очень ограниченной и неглубокой. Он может определять только следующие типы, распознаваемые JavaScript:

Boolean, string, bigint, symbol, undefined, function, number

  function example(x: number | string) {

      if (typeof x === 'string') {

          console.log(x.length); // x является string

      }

  }

**- instanceof: Проверяет принадлежность объекта к классу.**

instanceof — это встроенная защита типа, которую можно использовать для проверки того, является ли значение экземпляром заданной функции-конструктора или класса. С помощью этой защиты типа мы можем проверить, является ли объект или значение производным от класса, что полезно для определения типа экземпляра типа.

  class User {

      name: string;

  }

  function printName(user: User | null) {

      if (user instanceof User) {

          console.log(user.name); // user является User

      }

  }

**- in: Проверяет наличие свойства в объекте.**

Защита типа in проверяет, обладает ли объект определённым свойством, используя его для различения разных типов.

  interface A { x: number; }

  interface B { y: string; }

  function example(q: A | B) {

      if ('x' in q) {

          console.log(q.x); // q является A

      } else {

          console.log(q.y); // q является B

      }

  }

**Пользовательские тайпгарды**

Создание пользовательской защиты типов — это, как правило, самый мощный вариант использования защит типов. Когда вы создаёте пользовательскую защиту типов, написав её самостоятельно, нет никаких ограничений на то, что вы можете проверить. Однако если пользовательская защита типов написана неправильно, это может привести к множеству ошибок. Поэтому точность является ключевым фактором.

Пример пользовательского тайпгарда:

interface User {

    name: string;

}

function isUser(x: unknown): x is User {

    return typeof x === 'object' && x !== null && 'name' in x;

}

const user: unknown = { name: "Alice" };

if (isUser(user)) {

    console.log(user.name); // user является User

}

Пользовательские тайпгарды позволяют создавать более сложные проверки типов и делают код более безопасным и понятным.

**Заключение**

Приведение типов и тайпгарды являются важными инструментами в TypeScript, которые помогают разработчикам писать более безопасный и предсказуемый код. Приведение типов позволяет явно указывать типы переменных, а тайпгарды — управлять логикой сужения типов на основе условий. Использование этих механизмов способствует снижению количества ошибок и повышению качества кода.