JavaScript - дополнительно

Шаблонные строки, стили кода, JSDoc - документация, область видимости и устаревшая переменная VAR

Шаблонные строки

Шаблонные строки (template strings) в JavaScript представляют собой удобный способ вставки переменных и выражений в строки. Вместо использования конкатенации или сложных выражений с "+" или другими операторами, вы можете использовать синтаксис с обратными кавычками (`...`) для создания шаблонных строк.

Плюсы:

Интерполяция переменных: В шаблонных строках вы можете вставлять значения переменных напрямую внутри строки, используя синтаксис ${...}. Это делает код более читаемым и удобным для поддержки.

Многострочность: С шаблонными строками легко создавать многострочные строки, что особенно удобно для создания HTML-шаблонов или SQL-запросов.

Выражения: Вы можете вставлять в шаблонные строки любые JavaScript-выражения, включая вызовы функций, условные выражения и т. д.

Минусы:

Совместимость: Некоторые старые версии браузеров могут не поддерживать шаблонные строки, поэтому при написании кода нужно учитывать их совместимость.

Производительность: На небольших фрагментах кода использование шаблонных строк не повлияет на производительность, но на больших объемах итераций они могут замедлить исполнение из-за необходимости интерполяции выражений.

Применение:

Генерация HTML: Шаблонные строки часто используются для генерации HTML-кода в JavaScript, особенно в современных фронтенд-фреймворках.

Форматирование текста: Они удобны для форматирования текстовых сообщений, логов и вывода информации в консоль.

SQL-запросы: При работе с базами данных шаблонные строки могут использоваться для формирования SQL-запросов, облегчая вставку переменных и значений.

Локализация и интернационализация: Возможность вставлять переменные в строки делает их удобным инструментом при локализации и интернационализации приложений.

экранизация в шаблонной строке

Экранизация в шаблонных строках JavaScript позволяет вставлять специальные символы или выражения, которые были бы интерпретированы как часть синтаксиса шаблонных строк. Это особенно полезно, когда вы хотите вставить символы, такие как обратные кавычки (`), доллары ($) или обратные слеши ().

Обратные кавычки (`):

Если вам нужно вставить обратные кавычки внутрь шаблонной строки, вы можете их экранировать с помощью обратного слеша ().

```
const text = `\`Это экранированные обратные кавычки\``;
```

Доллар ($):

Для экранирования символа доллара используется обратный слеш.

```
const price = `\${10.00}`;
```

Обратный слеш ():

Если вам нужно вставить обратный слеш в шаблонную строку, используйте двойной обратный слеш (\).

```
const path = `C:\\Windows\\System32\\`;
```

Теговые шаблоны

это функциональность, которая позволяет обрабатывать шаблонные строки с помощью пользовательских функций, называемых "тегами". Это мощный инструмент, который позволяет создавать более гибкие и выразительные конструкции строк.

Преимущества:

- Гибкость: Теговые шаблоны позволяют создавать настраиваемые обработчики строк для различных целей.
- Читаемость: Позволяют создавать более выразительный и читаемый код.

Синтаксис

Теговый шаблон состоит из имени функции (тега), за которым следует шаблонная строка, заключенная в обратные кавычки (`...`). Функция (тег) получает два аргумента: массив строковых литералов и значения выражений, вставленных внутрь шаблона.

```
function myTag(strings, ...values) {
    // Обработка строк и значений
}

const result = myTag`Hello, ${name}!`;
```

Разбор примера

В этом примере мы определяем функцию greet, которая принимает два аргумента: массив строк strings и имя name. Затем мы используем теговый шаблон greet, чтобы передать строку и переменную username. Функция greet возвращает конкатенацию строк из массива strings с именем и последующими строками.

```
function greet(strings, name) {
    return `${strings[0]}${name}${strings[1]}`;
}

const username = 'Alice';
const greeting = greet`Привет, ${username}!`;

console.log(greeting); // "Привет, Alice!"
```

JavaScript разбивает шаблонную строку на массив строковых литералов (strings) и значения вставленных выражений (values). Массив strings содержит части строки, расположенные между вставленными значениями, включая начало и конец строки.

Стили кода

Стили кода - это соглашения о том, как форматировать и организовывать код, которые используются для обеспечения консистентности и читаемости в рамках проекта или сообщества разработчиков.

Плюсы:

- Читаемость: Стили кода способствуют созданию более читаемого и понятного кода, что делает его легче поддерживать и изменять в будущем.
- Консистентность: Установка единого стиля для всего проекта помогает предотвратить неразборчивые и противоречивые конструкции, что улучшает совместную работу в команде.
- Улучшенное обнаружение ошибок: Отслеживание стиля кода может помочь выявить потенциальные ошибки или проблемы в коде на ранних этапах разработки.

Минусы:

- Субъективность: Некоторые аспекты стилей кода могут быть субъективными и вызывать разногласия в команде.
- Изменение: Изменение стиля кода в уже существующем проекте может быть сложным и трудоемким процессом.

Особенности:

- Соглашения о форматировании: Это включает отступы, расстановку скобок, использование пробелов или табуляции и т. д.
- Именование: Это соглашения о выборе имен переменных, функций, классов и других элементов кода.
- Обработка ошибок: Соглашения о том, как обрабатывать ошибки и выбрасывать исключения.

стили именований

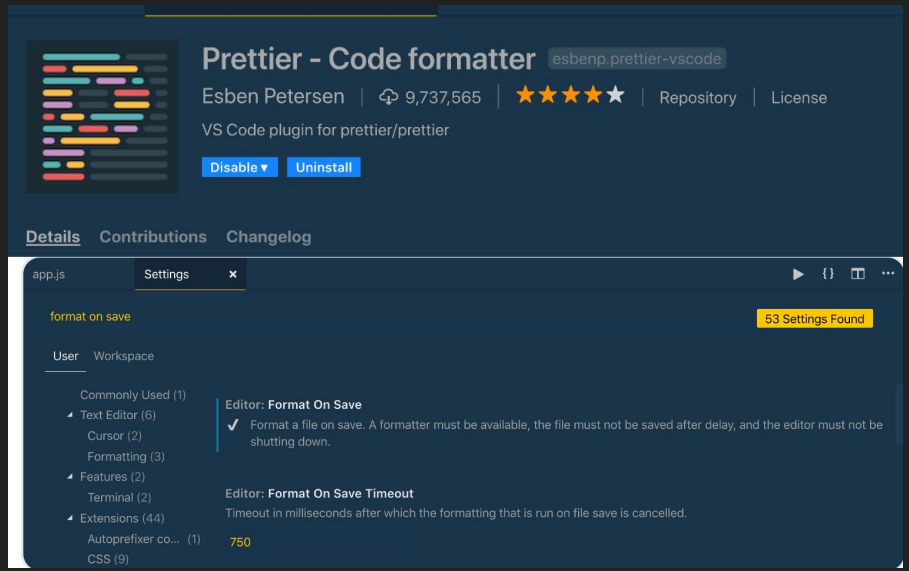
- Camel Case (верблюжий стиль) - myVariable, numberOfStudents
- Pascal Case - MyVariable, NumberOfStudents
- Snake Case (змеиный стиль) - my_variable, number_of_students
- Upper Snake Case - MY_CONSTANT, NUMBER_OF_STUDENTS
- Kebab Case (кебаб-кейс) - my-variable, number-of-students
- Lower Case - myvariable, numberofstudents

Форматирование кода

расширение для vs-code prettier

форматирование кода
CTRL + SHIFT + P =>
выбрать Format Document

форматирование при сохранении:



JSdoc

JSdoc - это способ документирования кода на JavaScript с использованием специального синтаксиса комментариев. Эти комментарии могут содержать описания функций, методов, классов и переменных, а также специальные теги для указания типов данных, параметров, возвращаемых значений и других деталей.

Плюсы:

Улучшение читаемости кода: Хорошо написанная документация делает код более понятным и помогает другим разработчикам быстрее разобраться в нем.

Автоматическая генерация документации: Инструменты, такие как JSDoc, могут использовать комментарии JSdoc для автоматической генерации документации в виде HTML-страниц или других форматов.

Поддержка в IDE: Многие интегрированные среды разработки (IDE) поддерживают JSdoc, предоставляя подсказки и автодополнение кода на основе комментариев.

Стандартизация: JSdoc имеет широкое распространение и широко используется в сообществе JavaScript, что делает его стандартом документирования.

Минусы:

Дополнительная нагрузка: Написание и поддержка документации может быть дополнительной нагрузкой для разработчика и может увеличить время разработки.

Возможность устаревания: Документация может устаревать, особенно если код часто изменяется, и не всегда поддерживается в актуальном состоянии.

пример

загрузить библиотеку для генерации документации

npm install -g jsdoc

сгенерировать документацию на основе jsdoc комментариев

jsdoc index.js

```
/**
 * Функция для вычисления суммы двух чисел.
 * @param {number} a - Первое число.
 * @param {number} b - Второе число.
 * @returns {number} - Сумма двух чисел.
 */
function add(a, b) {
    return a + b;
}
```

Область видимости

Область видимости в JavaScript определяет доступность переменных и функций в различных частях вашего кода. Это важная концепция, которая влияет на то, какие переменные можно использовать в определенном контексте и как они взаимодействуют друг с другом.

Области видимости в JavaScript-е

Глобальная область видимости (Global Scope):

- Переменные и функции, объявленные вне любых функций, находятся в глобальной области видимости.
- Они доступны из любой части вашего кода, включая другие функции.

Локальная область видимости (Local Scope):

- Переменные и функции, объявленные внутри функции, находятся в локальной области видимости этой функции.
- Они доступны только внутри этой функции и не видны извне.

Область видимости блока (Block Scope):

- Введенный в ECMAScript 6 (ES6), блочная область видимости позволяет ограничивать видимость переменных внутри блока кода, заключенного в фигурные скобки {}.
- Это, например, область видимости переменных, объявленных с помощью let и const.

Устаревшая переменная VAR

Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока.

Существует 2 основных отличия var от let/const:

- Переменные var не имеют блочной области видимости, они ограничены, как минимум, телом функции.
- Объявления (инициализация) переменных var производится в начале исполнения функции (или скрипта для глобальных переменных).

Ресурсы

npm пакет JSDoc - ТЫК

документация JSDoc - ТЫК

шаблонные строки документация - ТЫК

форматирование кода - ТЫК

документация prettier - ТЫК

область видимости в JS - ТЫК

область видимости для var - ТЫК