Кислый Илья 107а1

\\1

export interface User {

name: string;

age: number;

occupation: string;

}

export const users: User[] = [

{

name: 'Max Mustermann',

age: 25,

occupation: 'Chimney sweep'

},

{

name: 'Kate Müller',

age: 23,

occupation: 'Astronaut'

}

];

export function logPerson(user: User) {

console.log(`- ${user.name}, ${user.age}`);

}

console.log('Users:');

users.forEach(logPerson);

\\2

type TeacherCourse = Course & WithTeacherRole

type Teacher = User & WithLevel & {

courses: { [id: string]: TeacherCourse };

}

type Director = User & {

students: { [id: string]: User },

teachers: { [id: string]: Teacher & WithRate };

}

const s1: Student = {

id: "s1",

name: "s1",

courses: {

[1]: {

id: 1,

title: "First",

rate: 5,

role: "student",

level: "middle"

}

},

}

const t1: Teacher = {

id: "t1",

name: "t1",

level: "junior",

courses: {

[5]: {

id: 5,

title: "Fifth",

role: "teacher"

},

[1]: {

...s1.courses[1],

role: "teacher"

}

}

}

const d1: Director = {

id: "d1",

name: "d1",

students: {

["s1"]: s1,

["s2"]: {

id: "s2",

name: "s2"

}

},

teachers: {

["t1"]: {

...t1,

rate: 3,

},

["t2"]: {

id: "t2",

name: "t2",

level: "senior",

rate: 5

}

}

}

\\3

function zip<T, U>(first: T[], second: U[]): Array<[T, U]> {

const minLength = Math.min(first.length, second.length);

const result: Array<[T, U]> = [];

for (let i = 0; i < minLength; i++) {

result.push([first[i], second[i]]);

}

return result;

}

const q1: Array<[number, string]> = zip([1, 2, 3, 4, 5, 6], ["1", "2", "3"]);

const q2: Array<[boolean, boolean]> = zip([true], [false, false]);

console.log(q1, q2);

function groupBy<T, K, V>(source: T[], keySelector: (item: T, index: number) => K, valueSelector: (item: T, index: number) => V): Map<K, V[]> {

const result = new Map<K, V[]>();

for (let i = 0; i < source.length; i++) {

const item = source[i];

const key = keySelector(item, i);

const value = valueSelector(item, i);

if (!result.has(key)) {

result.set(key, []);

}

result.get(key)?.push(value);

}

return result;

}

const q3: Map<number, number[]> = groupBy([1, 2, 3, 4], x => x % 2, x => x + 1);

const q4: Map<boolean, {x: string, i: number}[]> = groupBy(["aaa", "bbb", "cc", "q", "lalaka"], (\_, i) => i % 2 === 0, (x, i) => ({i, x}));

console.log(q3, q4);

// 4

import { Omit, Readonly as RO, ReturnType } from 'some-library';

interface User {

id: number;

name: string;

age: number;

}

// Исключаем свойство 'age' из типа User

type UserWithoutAge = Omit<User, 'age'>;

// Преобразуем тип User в тип только для чтения (readonly)

type ReadonlyUser = RO<User>;

// Определяем тип возвращаемого значения функции

function greet(name: string): string {

return `Hello, ${name}!`;

}

type GreetReturnType = ReturnType<typeof greet>;

// Теперь мы можем использовать новые типы

const user: ReadonlyUser = { id: 1, name: 'Alice', age: 30 };

const newUser: UserWithoutAge = { id: 2, name: 'Bob' };

const greeting: GreetReturnType = greet('Charlie');

console.log(user, newUser, greeting);

\\ 5

document.addEventListener('DOMContentLoaded', () => {

const taskInput: HTMLInputElement | null = document.querySelector('#taskInput');

const taskList: HTMLUListElement | null = document.querySelector('#taskList');

function createTask() {

if (taskInput && taskList) {

const taskText: string = taskInput.value.trim();

if (taskText !== "") {

const currentDate: Date = new Date();

const dateString: string = currentDate.toLocaleDateString();

const newTask: HTMLLIElement = document.createElement("li");

newTask.innerHTML = `

<input type="checkbox" onchange="markAsDone(this)">

<span>${taskText} (${dateString})</span>

<button onclick="deleteTask(this)"><i class="far fa-times-circle"></i></button>

`;

taskList.appendChild(newTask);

taskInput.value = "";

} else {

alert("Пожалуйста, введите задачу!");

}

}

}

function markAsDone(checkbox: HTMLInputElement) {

const taskTextElement: HTMLElement | null = checkbox.nextElementSibling;

if (taskTextElement) {

if (checkbox.checked) {

taskTextElement.classList.add("done");

} else {

taskTextElement.classList.remove("done");

}

}

}

function deleteTask(button: HTMLButtonElement) {

const taskListItem: HTMLLIElement | null = button.parentNode as HTMLLIElement;

if (taskListItem) {

taskListItem.remove();

}

}

function clearAll() {

if (taskList) {

taskList.innerHTML = "";

}

}

const createTaskButton: HTMLButtonElement | null = document.querySelector('#createTaskButton');

const clearAllButton: HTMLButtonElement | null = document.querySelector('#clearAllButton');

if (createTaskButton) {

createTaskButton.addEventListener('click', createTask);

}

if (clearAllButton) {

clearAllButton.addEventListener('click', clearAll);

}

});