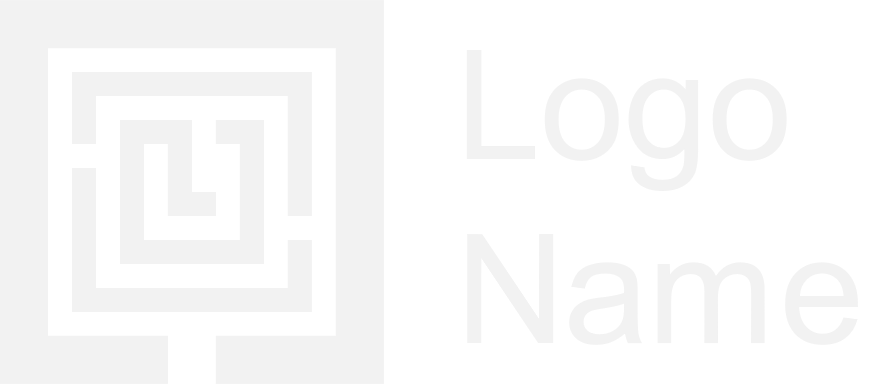
|  |
| --- |
| Typescript  doccument  Vue 3 |
| CLS-5 Street Address  City, ST ZIP Code  Phone  Email |



### **Tham khảo**

Tài liệu tham khảo:

1. <https://www.typescriptlang.org/docs/>
2. https://levunguyen.com/laptrinhjavascript/2021/03/04/khai-bao-bien-trong-typescript/

### **Lợi ích và khó khăn**

### **Cách sử dụng**

#### *Variable:*

*<scope>* <name>?: *<type> =* <value>

Ex: *const* props?: *object =* {

Prop1?: *‘hi’*

}

// khai báo biến trong vue 3 ts thông qua ref

*const* refForm = ref<props>({

Prop1?: *‘hi’*

})

#### *interface:*

##### Kiểu cơ bản

*interface* Props {

popperInlineEnd?: *Boolean | Number*

tag?: *string*

contentContainerTag?: *string*

isRtl?: *Boolean*

[propName: *string*]?: *any => Không quy định số lượng*

}

Ex:

*const* refForm = ref<Props>()

ex:

\\ ta có 1 đối tượng interface hoặc một đối tượng bí danh (aliases)

*type* objectAliases = {

Prop1: *string*

  Prop2: *number*

  Prop3: *boolean*

}

*interface* expArr {

  property?: *objectAliases*[]

=> biểu thị cho thuộc tính property là một mảng các đối tượng *objectAliases*

}

* Kiểu không quy định số lượng

*interface* myArray {

[propName: *string*]?: *any => Không quy định số lượng*

}

Ex:

let myArray: StringArray;

myArray = ["Bob", "Fred"];

* Kiểu lấy giá trị là key của một interface khác

*interface* Status {

  Verified: *string*

  Rejected: *string*

  Pending: *string*

}

*interface* Transition {

status: keyof Status

}

Ex:

*const* lastTransitions: Transition[] = [

  {

    status: 'Verified',

  },

  {

    status: 'Rejected',

  },

  {

    status: 'Pending',

  },

]

* Kiểu function

*interface* SearchFunc {

  (*source*: *string*, *subString*: *string*): *boolean*;

}

=>Kiểu property trên là kiểu function và giá trị output của function là Boolean hoặc void ,…

*let* mySearch: SearchFunc;

 mySearch = *function* (*source*: *string*, *subString*: *string*): *boolean* {

*let* result = *source*.search(*subString*);

   return result > -1;

 };

\*\*\*note: tích hợp vào emit

* Tham số chỉ đọc: readonly

Không thể thay đổi giá trị chỉ được phép đọc

interface Citizen {

name: string;

readonly SSN: number;

}

let personObj: Citizen = { SSN: 110555444, name: 'James Bond' }

personObj.name = 'Steve Smith'; // OK

personObj.SSN = '333666888'; // Compiler Error

* Kế thừa interface: Thường được dung cho các interface global cha

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | interface IPerson {  name: string;  gender: string;  }  interface IEmployee extends IPerson {  empCode: number;  }  let empObj:IEmployee = {  empCode:1,  name:"Bill",  gender:"Male"  } |

* Class kế thừ interface: implement interface

Cũng như các ngôn ngữ khác ts cũng hỗ trợ cho một class cài đặt interface

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | interface IEmployee {  empCode: number;  name: string;  getSalary:(number)=>number;  }  class Employee implements IEmployee {  empCode: number;  name: string;  constructor(code: number, name: string) {  this.empCode = code;  this.name = name;  }  getSalary(empCode:number):number {  return 20000;  }  }  let emp = new Employee(1, "Steve"); |

* Generic interface:

Ts interface cũng cho phép cài đặt generic

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | interface KeyPair<T, U> {  key: T;  value: U;  }  let kv1: KeyPair<number, string> = { key:1, value:"Steve" }; // OK  let kv2: KeyPair<number, number> = { key:1, value:12345 }; // OK |

Giúp linh hoạt kiểu dữ liệu hơn cho interface

#### *Function:*

*const* resetEvent : EventSourceFunc = (*info*, successCallback) *=>* {

  event.value = JSON.parse(JSON.stringify(props.event))

  nextTick(() *=>* {

    refForm.value?.resetValidation()

  })

}

##### **Tham số Rest:**

*Một function có thể có 1 hoặc nhiều tham số. Sẽ có những trường hợp ta không biết được sẽ truyền bao nhiêu tham số trong function. Để giải quyết vấn đề này TypeScript cung cấp cho chung ta tham số Rest với khai báo là 3 dấu … trong function.*

function Greet(greeting: string, ...names: string[]) {

return greeting + " " + names.join(", ") + "!";

}

Greet("Hello", "Steve", "Bill"); // returns "Hello Steve, Bill!"

Greet("Hello");// returns "Hello !"

L\*\*Lưu ý: tham số rest không được đặt ở tham số đầu tiên của function

|  |  |
| --- | --- |
| 1  2  3 | function Greet(...names: string[], greeting: string) { // Compiler Error  return greeting + " " + names.join(", ") + "!";  } |

#### *class:*

*Một Class thì gồm các phần như constructor, thuộc tính và phương thức.*

class Employee {

empCode: number;

empName: string;

constructor(code: number, name: string) {

this.empName = name;

this.empCode = code;

}

getSalary() : number {

return 10000;

}

}

let emp = new Employee(100,"Steve");

##### Kế thừa: **extends**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | class Person {  name: string;    constructor(name: string) {  this.name = name;  }  }  class Employee extends Person {  empCode: number;    constructor(empcode: number, name:string) {  super(name);  this.empCode = empcode;  }    displayName():void {  console.log("Name = " + this.name + ", Employee Code = " + this.empCode);  }  }  let emp = new Employee(100, "Bill");  emp.displayName(); // Name = Bill, Employee Code = 100 |

##### Cài đặt interface:

*Chúng ta có thể implements một hoặc nhiều Interface cho Class. Ví dụ Class Employee sau đây.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | interface IPerson {  name: string;  display():void;  }  interface IEmployee {  empCode: number;  }  class Employee implements IPerson, IEmployee {  empCode: number;  name: string;    constructor(empcode: number, name:string) {  this.empCode = empcode;  this.name = name;  }    display(): void {  console.log("Name = " + this.name + ", Employee Code = " + this.empCode);  }  }  let per:IPerson = new Employee(100, "Bill");  per.display(); // Name = Bill, Employee Code = 100  let emp:IEmployee = new Employee(100, "Bill");  emp.display(); //Compiler Error: Property 'display' does not exist on type 'IEmployee' |

##### Override method:

*Chúng ta có thể override lại method của lớp cha giống như trong Java.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | class Car {  name: string;    constructor(name: string) {  this.name = name;  }    run(speed:number = 0) {  console.log("A " + this.name + " is moving at " + speed + " mph!");  }  }  class Mercedes extends Car {    constructor(name: string) {  super(name);  }    run(speed = 150) {  console.log('A Mercedes started')  super.run(speed);  }  }  class Honda extends Car {    constructor(name: string) {  super(name);  }    run(speed = 100) {  console.log('A Honda started')  super.run(speed);  }  }  let mercObj = new Mercedes("Mercedes-Benz GLA");  let hondaObj = new Honda("Honda City")  mercObj.run(); // A Mercedes started A Mercedes-Benz GLA is moving at 150 mph!  hondaObj.run(); // A Honda started A Honda City is moving at 100 mph! |

##### Abstract class: Abstract

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | abstract class Person {  name: string;    constructor(name: string) {  this.name = name;  }  display(): void{  console.log(this.name);  }  abstract find(string): Person;  }  class Employee extends Person {  empCode: number;    constructor(name: string, code: number) {  super(name); // must call super()  this.empCode = code;  }  find(name:string): Person {  // execute AJAX request to find an employee from a db  return new Employee(name, 1);  }  }  let emp: Person = new Employee("James", 100);  emp.display(); //James  let emp2: Person = emp.find('Steve'); |

*Class mà extends một abstract class phải bắt buộc sử dụng từ khóa super trong constructor của mình.*

#### *export:*

export *const* useCalendar

export *const* useCalendar = (*event*: Ref<Event | NewEvent>) *=>* {

...

return {

    refCalendar,

    calendarOptions,

  }

}

#### *cycle hook:*

##### a. Computed:

*const* isAppRtl = computed({

   get() {

    return config.app.isRtl.value

    },

  set(*value*: typeof config.app.isRtl.value) {

    config.app.isRtl.value = *value*

    localStorage.setItem(`${config.app.title}-isRtl`, *value*.toString())

    \_setAppDir(*value* ? 'rtl' : 'ltr')

  },

})

*const* startDateTimePickerConfig = computed(() *=>* {

*const* config: Options = { enableTime: true, dateFormat: 'Y-m-d H:i' }

  if (event.value.end)

    config.maxDate = event.value.end

  return config

})

##### b. Mounted:

*const* updatePopper = async () *=>* {

...

}

onMounted(updatePopper)

 // 👉 onMounted

  onMounted(() *=>* {

    ...

  })

##### c. Watch:

<https://www.vuemastery.com/blog/vues-watch-vs-watcheffect-which-should-i-use/>

* Loại đơn

watch(prop, *val* *=>* {

. . .

  }, { immediate: true, deep: true })

* Loại kép

watch([prop1, prop2], *([newProp1, newProp2], [prevProp1, prevProp2])* *=>* {

. . .

stopWatch () => Dừng watchEffect HERE

  }, { immediate: true, deep: true })

##### d. WatchEffect:

*WatchEffect theo dõi toàn bộ sự thay đổi của component*

import { watchEffect } from 'vue'

watchEffect(() *=>* {

  // CODE HERE

stopWatchEffect() => Dừng watchEffect HERE

})

##### e. Setup: thay thế cho created

\*\*\*Note: - Khi không có template ta sẽ sử dụng dạng `optional api`

* Khi có template ta sẽ sử dụng composition api (setup lang=`ts`)

#### *key word:*

##### 1. StructuredClone: Clone một struct

structuredClone(data)

##### 2. Ref:

*Ref nhận một giá trị bên trong và trả về một đối tượng ref phản ứng và có thể thay đổi. Đối tượng ref có một thuộc tính duy nhất .value trỏ đến giá trị bên trong*

* Cách khai báo:

const count = ref(0);

* Muốn thay đổi giá trị của count ta phải thông qua ‘value’

count.value++;

* Ta có thể khai báo kiểu dữ liệu cho ref

ref<type>(value)

*type: có thể là interface, class, number, string, Boolean, Object, …*

* Muốn gọi sử dụng biến ref

Setup => return biến ref

return { count};

template:

<button @click="increaseCount">count is: {{ count }}</button>

*Ref hữu ích cho các nguyên hàm (chuỗi, số, boolean, v.v.)*

* Mở rộng:

*const* items = ['Item 1', 'Item 2', 'Item 3', 'Item 4'] as const

*const* select = ref<typeof items[number]>()

Quy định kiểu ref cho “select” là các loại được quy định trong mảng “items”

=> selected chỉ nhận các giá trị:

"Item 1" | "Item 2" | "Item 3" | "Item 4" | *undefined*

##### 3. Reactive:

*Phản ứng trả về một bản sao phản ứng của đối tượng. Chuyển đổi phản ứng là "deep"—nó ảnh hưởng đến tất cả các thuộc tính lồng nhau. Trong quá trình triển khai dựa trên*[*Proxy ES2015*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy)*, proxy được trả về không bằng với đối tượng ban đầu. Bạn nên làm việc độc quyền với proxy phản ứng và tránh dựa vào đối tượng ban đầu.*

* Cách khai báo:

const state = reactive({

count: 0

});

* Muốn thay đổi giá trị của count ta không cần phải thông qua ‘value’ như ref:

state.count++;

*reactive hữu ích cho các đối tượng và mảng.*

* Muốn gọi sử dụng biến reactive

Setup => return biến reactive

return { state};

template:

<button @click="increaseCount">count is: {{ state.count }}</button>

* Lưu ý:

*Reactive và ref có thể kết hợp lẫn nhau*

*Đối với ref khi ta truyền một đối tượng vào ref nó sẽ trả về một đối tượng đã được chuyển qua reactive*

const state = ref({

count: 0

});

Lúc này để thay đổi giá trị biến:

state.value.count++;

***\*\*Ngược lại để đổi từ reactive sang ref trong trường hợp sau***

* Đối tượng reactive:

const state = reactive({

count: 0

});

Khi ta muốn setup return các giá trị của đối tượng reactive thông qua detructoring => …state

return { ...state };

nhằm mục đích sử dụng

<button @click="increaseCount">count is: {{ count }}</button>

Thay vì state.count

Lúc này giá trị count sẽ không tồn tại, không thể trả giá trị của đối tượng reactive qua setup => để khắc phục ta thực hiện việc đổi reactive qua ref thông qua “toRef”

return { ...toRefs(state };

Lúc này trên template ta có thể sử dụng giá trị của reactive (count)

<button @click="increaseCount">count is: {{ count }}</button>

##### 4. Prop: defineProps

*const* props = withDefaults(defineProps<Type>(), {

=>Type kiểu dữ liệu cho prop thường là 1 interface

  property: ‘value’, value’=> giá trị mặc định cho prop

})

*Ex:*

*interface* Props {

  collapsed?: *boolean*

  noActions?: *boolean*

  actionCollapsed?: *boolean*

  actionRefresh?: *boolean*

  actionRemove?: *boolean*

  title?: *string*

}

*const* props = withDefaults(defineProps<Props>(), {

  collapsed: false,

  noActions: false,

  actionCollapsed: false,

  actionRefresh: false,

  actionRemove: false,

  title: undefined,

})

##### 5. Emit: defineEmits

*interface* Emit {

  (*e*: 'collapsed', *isContentCollapsed*: *boolean*): *void*

  (*e*: 'refresh', hideOverlay: () *=>* *void*): *void*

  (*e*: 'trash'): *void*

}

*const* emit = defineEmits<Emit>()

 emit('collapsed', isContentCollapsed.value)

 emit('refresh', hideOverlay)

 emit('trash')

or

defineEmits<{

  (*e*: 'change', *id*: *string*): *void*

}>()

##### 6. DefineExpose:

##### 7. ToRaw:

*Trả về một raw là nguyên bản của proxy do vue tạo ra*

*const* foo = reactive({ a: 1 })

Lúc này foo là một proxy: Proxy {a: 1}

Để lấy nguyên bản của nó là {a: 1}:

toRaw(foo) => {a:1}

Tương tự

*const* foo2 = ref({ a: 1 }) => nó là 1 ref và trả về 1 proxy thông qua value

toRaw(foo2.value) =>{a:1}

##### 8. Provide / inject:

* Sử dụng ở app global

app.provide('$msg', 'hello')

* Sử dụng cơ bản:

Tại nơi cung cấp dịch vụ:

import { provide } from 'vue'

provide('$msg', 'North Pole')

Tại nơi sử dụng dịch vụ:

import { inject } from 'vue'

const $msg = inject('$msg')

*Cấu hình nên property global*

##### 9. Tuple:

* *Sử dụng Tuple để chứa 2 hoặc nhiều giá trị có kiểu dữ liệu khác nhau*

|  |  |
| --- | --- |
| 1  2  3  4  5 | var empId: number = 1;  var empName: string = "Steve";  // Tuple type variable  var employee: [number, string] = [1, "Steve"];  var employee: [number, string][];  employee = [[1, "Steve"], [2, "Bill"], [3, "Jeff"]]; |

* Chúng ta truy cập mảng Tuple như sau:

|  |  |
| --- | --- |
| 1  2  3 | var employee: [number, string] = [1, "Steve"];  employee[0]; // returns 1  employee[1]; // returns "Steve" |

* Chúng ta sử dụng push để thêm phần tử cho Tuple.

|  |  |
| --- | --- |
| 1  2  3 | var employee: [number, string] = [1, "Steve"];  employee.push(2, "Bill");  console.log(employee); //Output: [1, 'Steve', 2, 'Bill'] |

##### 10. Union:

* Cho phép chúng ta sử dụng nhiều kiểu dữ liệu cho một biến

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | let code: (string | number);  code = 123; // OK  code = "ABC"; // OK  code = false; // Compiler Error  let empId: string | number;  empId = 111; // OK  empId = "E111"; // OK  empId = true; // Compiler Error |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | function displayType(code: (string | number))  {  if(typeof(code) === "number")  console.log('Code is number.')  else if(typeof(code) === "string")  console.log('Code is string.')  }  displayType(123); // Output: Code is number.  displayType("ABC"); // Output: Code is string.  displayType(true); //Compiler Error: Argument of type 'true' is not assignable to a parameter of type string | number |

##### 11. Any:

* Any sử dụng khi ta không biết kiểu dữ liệu là gì.
* Thường được sử dụng khi chúng ta gọi một webservice bên ngoài hoặc dịch vụ của nhà phát triển thứ 3.

|  |  |
| --- | --- |
| 1  2  3 | let something: any = "Hello World!";  something = 23;  something = true; |

**Chúng ta có thể tạo mảng với any như sau:**

|  |  |
| --- | --- |
| 1  2  3 | let arr: any[] = ["John", 212, true];  arr.push("Smith");  console.log(arr); //Output: [ 'John', 212, true, 'Smith' ] |

##### 12. Void:

* void được sử dụng để thông báo function không trả về kiểu dữ liệu gì.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function sayHi(): void {  console.log('Hi!')  }  let speech: void = sayHi();  console.log(speech); //Output: undefined |

##### 13. Never:

*Never có nghĩa giá trị đó sẽ không xảy ra. Never được sử dụng khi ta chắc chắn việc gì đó không xảy ra. Ví dụ sau chúng ta viết function nó sẽ không bao giờ trả về lỗi vì while luôn true nên vòng lặp while chạy vô hạn không bao giờ bị lỗi.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | function throwError(errorMsg: string): never {  throw new Error(errorMsg);  }  function keepProcessing(): never {  while (true) {  console.log('I always does something and never ends.')  }  } |

##### 14. Super:

class Car {

name: string;

constructor(name: string) {

this.name = name;

}

run(speed:number = 0) {

console.log("A " + this.name + " is moving at " + speed + " mph!");

}

}

*Để truy cập đến class cha từ class con để gọi hàm và biến ta sử dụng từ khoá super*

*Ta có thể kế thừ khởi tạo từ cha*

constructor(name: string) {

super(name);

}

Ta cũng có thể gọi method được cài đặt từ cha:

run(speed = 150) {

console.log('A Mercedes started')

super.run(speed);

}

##### 15. Type Aliases:

Cơ bản nó là một bí danh của một đối tượng, khi bạn muốn sử dụng ở nhiều nơi cho các loại cấu trúc phức tạp

<https://flow.org/en/docs/types/aliases/>

type MyObject<A, B, C, string, number > = {

foo: A,

bar: B,

baz: C,

property: string,  
method(): number,

};

var val: MyObject<number, boolean, string> = {

foo: 1,

bar: true,

baz: 'three',

};

export *type* EmailLabel = 'personal' | 'company' | 'important' | 'private'

#### *Quy tắc chuyển đổi dữ liệu:*

|  |
| --- |
| Typescript  doccument  Vue 3 |
| CLS-5 Street Address  City, ST ZIP Code  Phone  Email |

