

TypeScript 安装

1. 安装

- [安装指南](#)
 - [Windows](#)
 - [Mac OS](#)
 - [Linux](#)
 - [Docker](#)
- [安装 TypeScript](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 编译器](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 库](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 插件](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 配置](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 测试](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 文档](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 示例](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 工具](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 库](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 配置](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 测试](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 文档](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 示例](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)
- [安装 TypeScript 工具](#)
 - [使用 npm](#)
 - [使用 yarn](#)
 - [使用 Visual Studio Code](#)

安装 TypeScript 的步骤

TypeScript 安装 JavaScript 安装



JavaScript ██████████ TypeScript██████ 8████

- boolean
- string
- number
- bigint
- symbol
- object
- undefined
- null

TypeScript ███ JavaScript ██████████ 8████████ TypeScript █████

Number ☐ String ☐ Boolean ☐ JavaScript

 undefined null

8 TypeScript

boolean ☐

boolean

--	--	--	--	--

 true

--

 false

--	--	--	--	--

```
const x:boolean = true;  
const y:boolean = false;
```


 boolean

string

string

```
const x:string = 'hello';
const y:string = `${x} world`;
```

string

string

number

number

```
const x:number = 123;
const y:number = 3.14;
const z:number = 0xffff;
```

number

number

bigint

bigint

```
const x:bigint = 123n;
const y:bigint = 0xffffn;
```

bigint

x

y

bigint

bigint

```
const x:bigint = 123; //
const y:bigint = 3.14; //
```

bigint

bigint

ES2020

TypeScript

JavaScript

symbol

symbol

Symbol

```
const x:symbol = Symbol();
```

Symbol Symbol() symbol

symbol Symbol

object

JavaScript object

```
const x:object = { foo: 123 };
const y:object = [1, 2, 3];
const z:object = (n:number) => n + 1;
```

object

undefined null

undefined null

undefined undefined

```
let x:undefined = undefined;
```

x undefined undefined

null null

```
const x:null = null;
```

x null

strictNullChecks undefined null noImplicitAny any

```
// noImplicitAny strictNullChecks
```

```
let a = undefined; // any
const b = undefined; // any
```

```
let c = null; // any
const d = null; // any
```

strictNullChecks

```
// 严格模式 strictNullChecks
```

```
let a = undefined; // undefined
```

```
const b = undefined; // undefined
```

```
let c = null; // null
```

```
const d = null; // null
```

strictNullChecks

undefined

undefined

undefined

null

null

null

JavaScript

JavaScript

JavaScript 8

undefined

null

object

primitive value

primitive value

- boolean
- string
- number
- bigint
- symbol

wrapper object

“ ”

“ ”

```
'hello'.charAt(1) // 'e'
```

JavaScript

hello

charAt()

JavaScript

JavaScript

JavaScript

charAt()

JavaScript

symbol

symbol

bigint

Symbol()

BigInt()

JavaScript

- Boolean()
- String()
- Number()

[illegible]

```
const s = new String('hello');
typeof s // 'object'
s.charAt(1) // 'e'
```

Diagram illustrating memory layout for variables and their values:

- `s` (5 bytes) contains the string `hello`.
- `hello` (6 bytes) contains the string `hello`.
- `typeof` (7 bytes) contains the string `object`.
- `object` (7 bytes) contains the string `string`.
- `string` (7 bytes) contains the string `string`.

The total size of the memory layout is 32 bytes.

The diagram illustrates a memory stack with three frames. The top frame is labeled 'new' and contains a 'String()' object and a 'new' label. The middle frame contains a 'new' label. The bottom frame contains a 'Number()' object and a 'Boolean()' object.

[illegible]

```
'hello' // []  
new String('hello') // []
```

□□□□□□□□ TypeScript □□□□□□□□□□□□□□□□□□□□

- Boolean $\mapsto$ boolean
- String $\mapsto$ string
- Number $\mapsto$ number
- BigInt $\mapsto$ bigint
- Symbol $\mapsto$ symbol

[illegible]

```
const s1:String = 'hello'; // OK
const s2:String = new String('hello'); // OK

const s3:string = 'hello'; // OK
const s4:string = new String('hello'); // OK
```

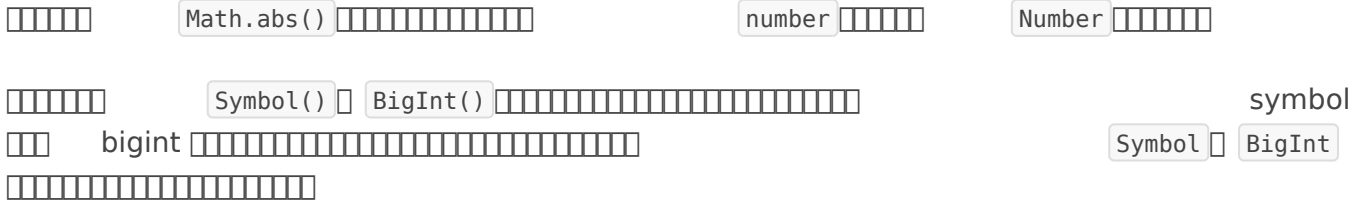
```

graph LR
    String[String] --> string[string]
    string --> array[Hello, World!]
  
```

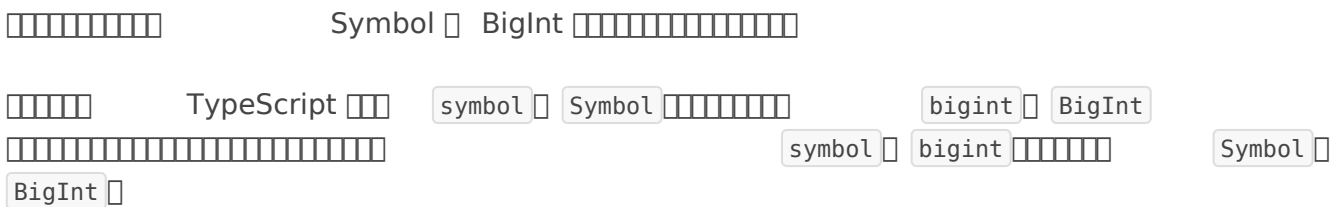
The diagram illustrates a memory layout. The top row consists of 32 small, empty rectangular slots arranged horizontally. Below this, the text "TypeScript" is centered. Underneath the text, there are four small, empty rectangular slots arranged horizontally.

```
const n1:number = 1;
const n2:Number = 1;

Math.abs(n1) // 1
Math.abs(n2) // 1
```



```
let a = Object(Symbol());
let b = Object(BigInt());
```



Object object

TypeScript Object object

Object

Object JavaScript Object

```
let obj:Object;

obj = true;
obj = 'hi';
obj = 1;
obj = { foo: 123 };
obj = [1, 2];
obj = (a:number) => a + 1;
```



undefined null Object

```
let obj:Object;

obj = undefined; // 
obj = null; // 
```

undefined null Object

{ } Object Object

```
let obj:{};

obj = true;
obj = 'hi';
obj = 1;
obj = { foo: 123 };
obj = [1, 2];
obj = (a:number) => a + 1;
```

obj { } Object

Object

object

object JavaScript

```
let obj:object;

obj = { foo: 123 };
obj = [1, 2];
obj = (a:number) => a + 1;
obj = true; // 
obj = 'hi'; // 
obj = 1; // 
```

object

object Object

Object

object

JavaScript

```
const o1:Object = { foo: 0 };
const o2:object = { foo: 0 };

o1.toString() // 
o1.foo // 

o2.toString() // 
o2.foo // 
```

toString()

foo

undefined null

undefined null

undefined null

```
let age:number = 24;

age = null;      // 
age = undefined; // 
```

age

number

null

undefined

undefined

null

number

undefined

null

JavaScript

JavaScript

undefined

null

TypeScript

```
const obj:object = undefined;
obj.toString() // 
```

obj

undefined

obj.toString()

undefined

TypeScript `strictNullChecks`

`undefined` `null` `any` `unknown`

`tsc`

```
// tsc --strictNullChecks app.ts
```

```
let age:number = 24;
```

```
age = null; //
```

```
age = undefined; //
```

`--strictNullChecks` `number` `age` `undefined` `null`

`tsconfig.json`

```
{
  "compilerOptions": {
    "strictNullChecks": true
    // ...
  }
}
```

`strictNullChecks` `undefined` `null`

```
// strictNullChecks
```

```
let x:undefined = null; //
```

```
let y:null = undefined; //
```

`undefined` `null` `null` `undefined`

`strictNullChecks` `undefined` `null` `any` `unknown`

```
let x:any = undefined;
```

```
let y:unknown = null;
```



TypeScript

“ ”

```
let x:'hello';  
  
x = 'hello'; //  
x = 'world'; //
```

x hello

TypeScript const

```
// x "https"  
const x = 'https';  
  
// y string  
const y:string = 'https';
```

x const TypeScript https string

const

const

```
// x { foo: number }  
const x = { foo: 1 };
```

x foo number JavaScript const

```
const x:5 = 4 + 1; //
```

5 4 + 1 TypeScript number 5
number number 5

```
let x:5 = 5;  
let y:number = 4 + 1;
```

```
x = y; // []
y = x; // []
```

[][][]

x []

y []

x []

y []

[]

```
const x:5 = (4 + 1) as 5; // []
```

[][][]

4 + 1 []

as 5 []

4 + 1 []

5

[][][]

[][][][][][][][][][][][][][][][][][]

[][][]

union types []

| []

[][]

A|B []

A [] B []

A|B []

```
let x:string|number;
```

```
x = 123; // []
```

```
x = 'abc'; // []
```

[][][]

x []

string|number []

[][][][][][][][][][][][][][][][]

```
let setting:true|false;
```

```
let gender:'male' | 'female';
```

```
let rainbowColor:'' | '' | '' | '' | '' | '' | '' | '';
```

[][][][][][][][][][][][][][][][]

true|false

[][][]

boolean []

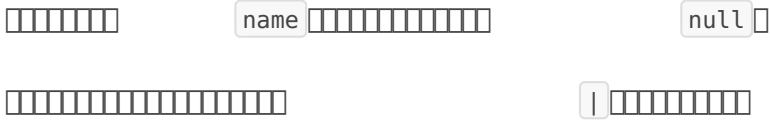
[][][]

strictNullChecks []

undefined [] null

[][][][][][][][][][][][][][]

```
name = 'John';  
name = null;
```



one



```
id.toUpperCase()
```



id toUpperCase()

“” TypeScript
“” type widening “” type narrowing
“” “”

```
function getPort(  
  scheme: 'http'|'https'  
) {  
  switch (scheme) {  
    case 'http':  
      return 80;  
    case 'https':  
      return 443;  
  }  
}
```

scheme



intersection types &
A&B A B A&B A B

```
let x:number&string;
```

x TypeScript x never



```
let obj:  
  { foo: string } &  
  { bar: string };  
  
obj = {  
  foo: 'hello',  
  bar: 'world'  
};
```

obj

foo

bar

```

type A = { foo: number };

type B = A & { bar: number };

```

B

A

bar

type

type

```

type Age = number;

let age:Age = 55;

```

type

number

Age

number

Age

```

type Color = 'red';
type Color = 'blue'; // 

```

Color

```

type Color = 'red';

if (Math.random() < 0.5) {
  type Color = 'blue';
}

```

if

Color

Color

```
type World = "world";  
type Greeting = `hello ${World}`;
```

```

graph LR
    Obj[Object] -- Greeting --> Greeting[00000000: Hello]
    Obj -- World --> World[00000004: World]
    Obj -- type --> Type[0000000C: JavaScript]
    
```

```
typedef struct {
```

JavaScript typeof

```
typeof 'foo'; // 'string'
```

`int` `sizeof` `int` `sizeof` `int` `sizeof` `int` `sizeof` `int`

```

[ ][ ][ ][ ]      sizeof [ ][ ][ ][ ][ ][ ][ ][ ]

```

JavaScript typeof

```
typeof undefined; // "undefined"
typeof true; // "boolean"
typeof 1337; // "number"
typeof "foo"; // "string"
typeof {}; // "object"
typeof parseInt; // "function"
typeof Symbol(); // "symbol"
typeof 127n // "bigint"
```

□□□□ typeof □□□ JavaScript □□□□□□□□□□□□

TypeScript

TypeScript 

```
const a = { x: 0 };

type T0 = typeof a;    // { x: number }
type T1 = typeof a.x;  // number
```

The diagram illustrates the TypeScript compiler's process of inferring the type of a variable 'a'. It shows a sequence of steps: first, the variable 'a' is declared with the type 'TypeScript'. Then, a value is assigned to 'a' using the expression '{ x: number }'. Finally, the type of 'a' is inferred to be 'number'.

```

a: TypeScript
{ x: number }
a: number

```

typeof

TypeScript

typeof

JavaScript

TypeScript

```

let a = 1;
let b:typeof a;

if (typeof a === 'number') {
  b = a;
}
```

typeof

JavaScript
typeof
JavaScript
TypeScript
typeof
TypeScript

```

let a = 1;
let b;
if (typeof a === 'number') {
  b = a;
}
```

typeof
typeof

JavaScript
TypeScript
typeof

```

type T = typeof Date(); //
```

typeof
Date()

typeof

```

type Age = number;
type MyAge = typeof Age; //
```

Age

typeof

typeof 


TypeScript 

foo 

typeof foo



TypeScript 

```
if (true) {  
  type T = number;  
  let v:T = 5;  
} else {  
  type T = string;  
  let v:T = 'hello';  
}
```



T



TypeScript 

```
type T = number|string;  
  
let a:number = 1;  
let b:T = a;
```



a



a



b



b



a



TypeScript 



subtype

number

A

B

A

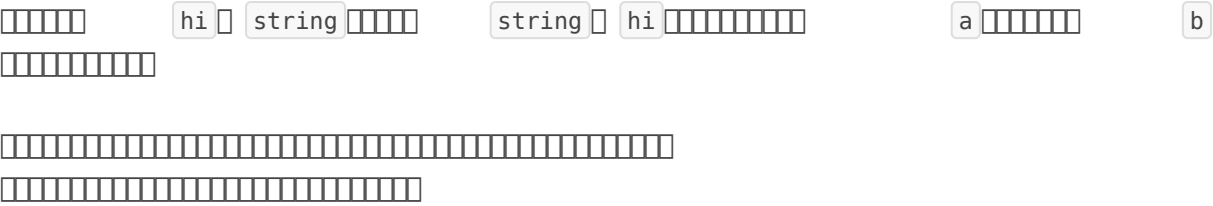
B

number|string

TypeScript 

```
let a:'hi' = 'hi';  
let b:string = 'hello';  
  
b = a; //
```

```
a = b; // 
```



```
#2
2023-07-09 12:07:22 UTC
2026-06-22 09:23:27 UTC
```