

01

Vue3+TS开发环境创建

基于Vite创建Vue3 + TS环境

vite除了支持基础阶段的纯TS环境之外，还支持 Vue + TS开发环境的快速创建, 命令如下：

```
npm create vite@latest vue-ts-pro -- --template vue-ts
```

说明：

1. npm create vite@latest 基于最新版本的vite进行项目创建
2. vue-ts-pro 项目名称
3. -- --template vue-ts 选择Vue + TS的开发模板

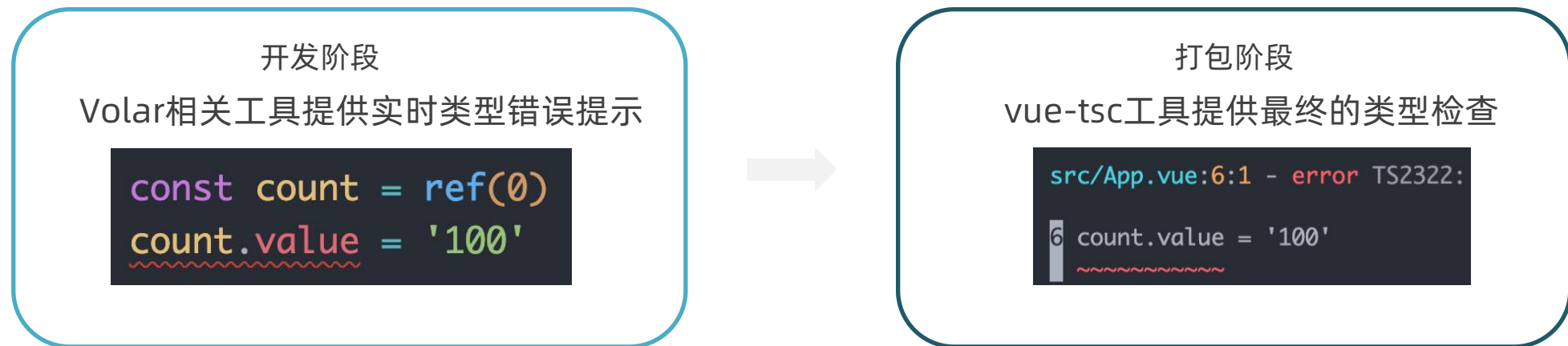
和.vue文件TS环境相关的工具职责说明

开发阶段

1. **Volar工具**对.vue文件进行实时的类型错误反馈
2. **TypeScript Vue Plugin** 工具用于支持在 TS 中 import *.vue 文件

打包阶段

vue-tsc工具负责打包时最终的类型检查



好处：开发阶段的类型提示交给IDE做，**保证vite的运行速度**，打包阶段做‘兜底类型校验’，确保类型无误



02

为ref标注类型

场景和好处

为ref标注类型之后，既可以在给ref对象的value赋值时校验数据类型，同时在使用value的时候可以获得代码提示

```
type ListItem = {  
  id: number  
  name: string  
}  
const list = ref<ListItem[]>([])  
list.value = [{  
  id: 100,  
  name: 'jack'  
}]
```

```
<template>  
  <ul>  
    <li v-for="item in list" :key="item."></li>  
  </ul>  
</template>
```

- id
- name

说明：本质上是给ref对象的value属性添加类型约束

如何标注类型

ref 函数和 TS 的配合通常分为两种情况，**类型推导**和**泛型指定类型**

1. 如果是简单的数据，推荐使用**类型推导**

```
const count: Ref<number>  
const count = ref(100)
```

2. 如果是较复杂的数据，通过**泛型指定类型**

```
const year = ref<string | number>(2028)  
  
year.value = '2029'  
year.value = 2030
```

说明：通过泛型指定类型之后除了修改value做了类型约束，ref函数的初始值也有了类型约束

思考

标注ref函数类型，可以满足把下图所示的数据赋值给value属性

```
1 [{  
2   id: '1001',  
3   name: '男士鞋子',  
4   price: 280,  
5 }, {  
6   id: '1002',  
7   name: '男士卫衣',  
8   price: 180,  
9 }]
```



03

为reactive标注类型

场景和好处

为reactive标注类型之后，既可以在响应式对象在修改属性值的时候约束类型，也可以在使用时获得代码提示

```
const form = reactive({  
  username: '',  
  password: ''  
})  
form.username = '100'  
form.username = 100
```

```
<template>  
  {{form.}}  
</templat  password  
            username
```

如何标注类型

reactive函数和TS的配合通常分为两种情况，**类型推导和显式标注**

1. 如果根据**默认参数对象推导的类型符合要求**，推荐使用**类型推导**

```
const form: {  
  username: string;  
  password: string;  
}  
  
const form = reactive({  
  username: '',  
  password: ''  
})
```

2. 如果根据默认对象**推导不出我们想要的类型**，推荐使用**类型别名**给变量显式注解对应类型

```
const form: {  
  username: string  
  password: string  
  isAgree?: boolean  
}
```

```
type Form = {  
  username: string  
  password: string  
  isAgree?: boolean  
}
```

```
const form: Form = reactive({  
  username: '',  
  password: ''  
})
```

说明：显式注解变量之后除了对**属性赋值的时候做了类型约束**而且要求**reactive的初始值**也需要满足类型要求



思考

标注reactive函数类型，可以满足下图所示数据格式的赋值操作
其中gender为选填字段

```
1 {  
2   username: 'jack',  
3   address: '北京市朝阳区',  
4   phone: '1321111111',  
5   gender: '男'  
6 }
```



03

为computed标注类型

如何进行类型标注

计算属性通常由已知的响应式数据计算得到，所以依赖的数据类型一旦确定通过自动推导就可以知道计算属性的类型

另外根据最佳实践，计算属性多数情况下是只读的，不做修改，所以配合TS一般只做代码提示

```
const count = ref(0)

const doubleCount: ComputedRef<number>
const doubleCount = computed(() => count.value * 2)
```

更复杂的例子

需求：给ref函数标注类型，接收后端返回的对象列表，然后使用计算属性做过滤计算，计算得到单价大于500的商品

```
1  [  
2    { id: '1001', name: '男鞋', price: 888 },  
3    { id: '1002', name: '女鞋', price: 555 },  
4    { id: '1002', name: '衬衫', price: 400 }  
5  ]
```

03

为事件处理函数标注类型

为什么事件处理函数需要标注类型

原生dom事件处理函数的参数默认会自动标注为any类型，没有任何类型提示，为了获得良好的类型提示，需要手动标注类型

```
1 <script setup lang="ts">
2 const inputChange = (e) => {
3   // e自动推断为any类型
4 }
5 </script>
6
7 <template>
8   <div>
9     <input type="text" @change="inputChange">
10   </div>
11 </template>
```


如何为事件处理函数标注类型

事件处理函数的类型标注主要做俩个事

1. 给事件对象形参 e 标注为 **Event** 类型，可以获得事件对象的相关类型提示

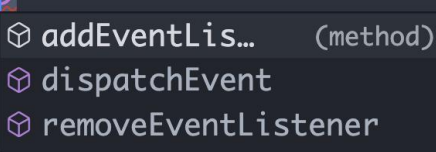
```
const inputChange = (e: Event) => {  
  console.log(e.)  
}  
  
</script>  
<template>
```



- defaultPrevented
- eventPhase
- isTrusted
- preventDefault
- stopImmediatePropagation
- stopPropagation
- target

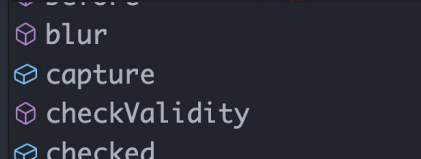
2. 如果需要更加精确的DOM类型提示可以使用 **断言 (as)** 进行操作

```
const inputChange = (e: Event) => {  
  console.log(e.target.)  
}  
  
</script>
```



- addEventListener (method)
- dispatchEvent
- removeEventListener

```
const inputChange = (e: Event) => {  
  console.log((e.target as HTMLInputElement).)  
}  
  
</script>
```



- blur
- capture
- checkValidity
- checked



思考

思考一下如何给一个按钮元素button的点击事件函数标注类型?

04

为模版引用标注类型

为什么给模板引用标注类型

给模版引用标注类型，本质上是给ref对象的value属性添加了类型约束，约定value属性中存放的是特定类型的DOM对象，从而在使用的时候获得相应的代码提示

```
1 <script setup>
2 import { onMounted, ref } from 'vue'
3 const elRef = ref<HTMLElement>(null)
4 onMounted(() => elRef.value.focus())
5 </script>
6
7 <template>
8   <div>
9     <input type="text" ref="elRef">
10   </div>
11 </template>
```

如何进行类型标注

通过具体的DOM类型联合null做为泛型参数, 比如我们想获取一个input dom元素

```
1 <script setup lang="ts">
2 import { onMounted, ref } from 'vue'
3
4 const inputRef = ref<HTMLInputElement | null>(null)
5
6 onMounted(() => {
7   inputRef.value?.focus()
8 })
9
10 </script>
11
12 <template>
13   <div>
14     <input type="text" ref="inputRef">
15   </div>
16 </template>
```

说明:

1. 泛型参数中通过将来获取dom的实际类型联合null进行类型标注
2. 因为在组件完全挂载完毕之前ref的值都是null, 所以需要在访问ref.value时使用可选链防止类型错误



思考

尝试为模版引用标注类型获取一个a元素?

05

对象的非空值处理

空值场景说明

当对象的属性可能是 `null` 或 `undefined` 的时候，称之为“空值”，尝试访问空值身上的属性或者方法会发生类型错误

```
const inputRef = ref<HTMLInputElement | null>(null)

onMounted(() => {
  inputRef.value.focus()
})
```

说明：inputRef变量在组件挂载完毕之前，value属性中存放的值为null，不是input dom对象，通不过类型校验

可选链方案

可选链 `?.` 是一种访问嵌套对象属性的安全的方式, 可选链前面的值为 `undefined` 或者 `null` 时, 它会停止运算

```
const inputRef = ref<HTMLInputElement | null>(null)

onMounted(() => {
  inputRef.value?.focus()
})
```

逻辑判断方案

通过逻辑判断，只有有值的时候才继续执行后面的属性访问语句

```
const inputRef = ref<HTMLInputElement | null>(null)

onMounted(() => {
  if (inputRef.value) {
    inputRef.value.focus()
  }
})
```

非空断言方案

非空断言 (!) 是指我们开发者明确知道当前的值一定不是null或者undefined，让TS通过类型校验

```
const inputRef = ref<HTMLInputElement | null>(null)

onMounted(() => {
  inputRef.value!.focus()
})
```

注意：使用非空断言要格外小心，它没有实际的JS判断逻辑，只是通过了TS的类型校验，容易直接把空值出现在实际的执行环境里



总结

1. TS里出现的“空值”场景指的是什么?

对象属性值为null或者undefined的场景

2. 非空断言 (!) 和可选链/逻辑判断相比需要注意什么?

确保对象属性一定不能为空值

05

为组件的props标注类型

为什么给props标注类型

1. 确保给组件传递的prop是类型安全的

```
<Button color="red"></Button>  
<Button :color="100"></Button>
```

2. 在组件内部使用props和为组件传递prop属性的时候会有良好的代码提示

```
props.
```

- color
- size?

```
<Button color="red" bt></Button>
```

- btn-type
- btn

props类型标注基础使用

语法：通过defineProps宏函数对组件props进行类型标注

需求：按钮组件有俩个prop参数，color类型为string且为必填，size类型为string且为可选，怎么定义类型？

```
1 // 1. 使用别名类型或者接口定义Props类型
2 type Props = {
3   color: string
4   size?: string
5 }
6
7 // 2. 使用defineProps注解类型
8 const props = defineProps<Props>()
```

说明：按钮组件传递prop属性的时候必须满足color是必传项且类型为string, size为可选属性，类型为string

props默认值设置

场景：Props中的可选参数通常除了指定类型之外还需要提供默认值，可以使用withDefaults宏函数来进行设置

需求：按钮组件的size属性的默认值设置为 middle

```
1 <script setup lang="ts">
2
3 type Props = {
4   color: string
5   size?: string
6 }
7 const props = withDefaults(defineProps<Props>(), {
8   size: 'middle'
9 })
10
11 </script>
```

说明：如果用户传递了size属性，按照传递的数据来，如果没有传递，则size值为 'middle'



总结

1. 为props标注类型的两个步骤是什么?

1. 通过type定义Props对象类型

2. 通过defineProps<>() 泛型传参

2. 为props设置默认值用哪个宏函数?

withDefaults(defineProps<Props>(), { 默认值设置 })



思考

给按钮组件添加一个btnType属性，类型为 'success'，
'danger' 或者 'warning' 三选一，默认值为 'success'



06

为组件的emits标注类型

为什么给组件的emits标注类型

问题：触发一个自定义事件，最需要关注的是什么？

```
1  const clickHandler = () => {  
2    emit('get-msg', 'i am son')  
3  }  
    事件名称      事件参数
```

作用：可以约束事件名称并给出自动提示，确保不会拼写错误，同时约束传参类型，不会发生参数类型错误

如何为组件的emits标注类型

语法：通过 `defineEmits` 宏函数进行类型标注

需求：子组件触发一个名称为 'get-msg' 的事件，并且传递一个类型为string的参数

```
1 // 1. 定义事件类型Emits
2 type Emits = {
3   (e: 'get-msg', msg: string): void
4 }
5
6 // 2. 给泛型参数传参
7 const emit = defineEmits<Emits>()
```

```
1 emit('get-msg', 'i am son')
```



总结

1. 为组件emits标注类型约束了哪俩个地方?

事件名称 + 参数类型

2. 为组件emits标注类型的俩个步骤是什么?

1. 通过type定义Emits类型

2. 通过defineEmits<Emits>() 泛型传参



思考

Son组件再触发一个事件'get-list', 传递参数类型为下图所示的数据类型

```
1  [  
2    {  
3      id: 1001,  
4      name: '冬季棉袜'  
5    }  
6  ]
```



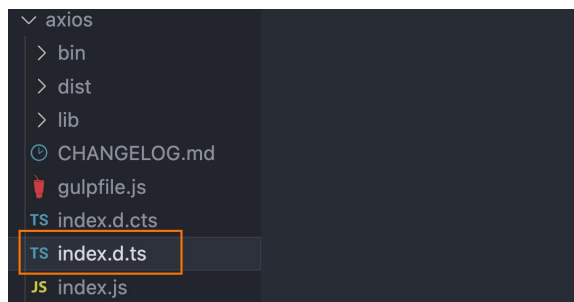
07

类型声明文件

什么是类型声明文件

概念：在TS中以`d.ts`为后缀的文件就是类型声明文件，主要作用是**为js模块提供类型信息支持**，从而获得类型提示

```
import axios from 'axios'
axios. 应为标识符。
```



说明：

1. d.ts是如何生效的？

在使用js某些模块的时候，**TS会自动导入模块对应的d.ts文件**，以提供类型提示

2. d.ts是怎么来的？

库如果本身是使用TS编写的，在打包的时候**经过配置自动生成对应的d.ts文件**（axios本身就是TS编写的）

使用 DefinitelyTyped 提供类型声明文件

场景：有些库本身并不是采用TS编写的，无法直接生成配套的d.ts文件，但是也想获得类型提示，此时需要 Definitely Typed 提供类型声明文件

```
import jquery from 'jquery'
```

已声明“jquery”，但从未读取其值。
无法找到模块“jquery”的声明文件。

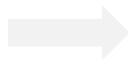
DefinitelyTyped是一个TS类型定义的仓库，专门为JS编写的库可以提供类型声明，比如可以安装 @types/jquery 为 jquery提供类型提示

```
import jquery from 'jquery'  
(alias) const jquery: JQueryStatic  
import jquery
```

TS内置类型声明文件

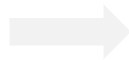
TS为JS运行时可用的所有标准化内置API都提供了声明文件，这些文件既不需要编译生成，也不需要三方提供

```
JS test.js 1 ●
src > JS test.js > ...
1  const list = []
2  list.
  entries
  every
  fill
  filter
```



```
TS lib.es5.d.ts ×
ders > qz > d5p9940n01jgllkpxnj12wr40000g
/**
```

```
JS test.js ●
src > JS test.js
1  document.ge
  getAnimati... (method) Do
  getElementById
  getElementsByName
  getElementsByClassName
```



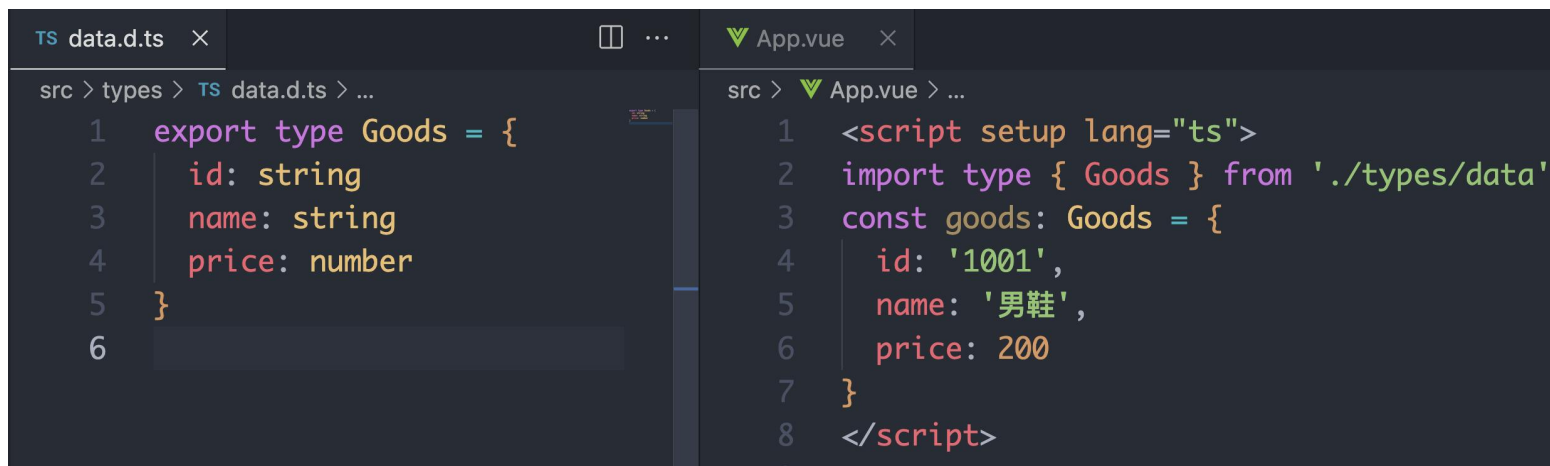
```
TS lib.dom.d.ts ×
lders > qz > d5p9940n01jgllkpxnj12wr40000
```

说明：这里的lib.es5.d.ts以及lib.dom.d.ts都是内置的类型声明文件，为原生js和浏览器API提供类型提示

自定义类型声明文件

d.ts文件在项目中是可以进行自定义创建的，通常有两作用，第一个是**共享TS类型（重要）**，第二种是给js文件提供类型（了解）

场景一：共享TS类型



```
TS data.d.ts ×  App.vue ×
src > types > TS data.d.ts > ...
1 export type Goods = {
2   id: string
3   name: string
4   price: number
5 }
6

src > App.vue > ...
1 <script setup lang="ts">
2 import type { Goods } from './types/data'
3 const goods: Goods = {
4   id: '1001',
5   name: '男鞋',
6   price: 200
7 }
8 </script>
```

说明：哪个业务组件需要用到类型导入即可，**为了区分普通模块，可以加上type关键词**

自定义类型声明文件

场景二：给JS文件提供类型

```
JS index.js ×  
src > add > JS index.js > ...  
1  const add = (a, b) => a + b  
2  export { add }  
3
```

```
TS index.d.ts ×  
src > add > TS index.d.ts > ...  
1  declare const add: (a: number, b: number) => number  
2  export { add }  
3
```

说明：通过`declare`关键词可以为js文件中的变量声明对应类型，这样js导出的模块在使用的时候也会获得类型提示

.ts文件和d.ts文件对比

TS中有俩种文件类型，一种是.ts文件，一种是.d.ts文件

.ts文件

1. 既可以包含类型信息也可以写逻辑代码
2. 可以被编译为js文件

.d.ts文件

1. 只能包含类型信息不可以写逻辑代码
2. 不会被编译为js文件，仅做类型校验检查



总结

1. d.ts的作用是什么?

为js模块提供类型信息

2. 使用原生的JS数组方法为什么会有类型提示?

TS内置了标准API相关的d.ts文件

3. d.ts文件里可以编写可运行的ts代码吗?

不可以，仅提供类型信息的代码

4. d.ts文件在业务中的场景是什么?

业务中共享类型

08

Vue3 +TS 综合案例

需求描述



功能描述

1. 频道列表渲染实现
2. 文章列表渲染实现
3. 点击频道切换对应的文章列表数据

技术方案

Vue3 + TypeScript + Axios + Vant

克隆项目

项目的初始化配置**已经完成**，其中包括：

1. 基于Vite搭建的基础的 Vue3 + TS 开发环境
2. 请求库 axios
3. 移动端组件库Vant
3. 静态结构组件模板

以下是项目地址，直接克隆项目安装依赖并run起来

```
git clone http://git.itcast.cn/heimaqianduan/vue3-ts.git
```

频道列表渲染 - 类型思想转变

之前业务开发我们用的是JavaScript，现在要加上TypeScript的类型，该如何把类型加进来呢？

核心业务实现逻辑

1. 使用ref创建响应式列表
2. 使用axios获取后端数据
3. 把后端数据赋值给响应式列表
4. 使用响应式列表渲染模版

添加类型之后

1. 存放后端数据 - 根据后端接口定义ref类型
2. 获取后端数据 - 根据后端接口定义axios返回的数据类型
3. 1/2依赖于相同的后端接口定义类型，赋值自然类型匹配
4. 享受TS类型带来的好处（类型安全 + 良好提示）

核心思想：使用TS之后的业务开发思想是保持一致的，重要的是根据接口格式定义响应式数据的类型以及axios返回数据的类型即可

频道列表渲染 - 定义axios的返回数据类型

定义axios的返回数据类型需要配合一个axios的request方法通过泛型指定

```
1 axios.request<根据后端接口定义的数据类型>
```

| | |
|------------|-------------------|
| object {2} | |
| ▼ data | object {1} |
| ▼ channels | array[object] {2} |
| id | integer |
| name | string |
| message | string |

```
1 // 类型定义
2 type ChannelItem = {
3   id: number
4   name: string
5 }
6
7 type ChannelRes = {
8   data: {
9     channels: ChannelItem[]
10  }
11   message: string
12 }
```

```
const getChannellist = async () => {
  const res = await axios.request<ChannelRes>({
    url: 'http://geek.itheima.net/v1_0/channels',
  })
  console.log(res.data)
}
```

data (property) data

message

说明：我们通过泛型参数传给request方法的ChannelRes类型约束了axios返回值res的data属性的类型

文章列表渲染实现



基础文章列表实现（固定频道id）

1. 根据接口格式定义数据类型
2. 使用ref 函数定义响应式数据
3. 使用 axios 请求数据并赋值给响应式数据
4. 数据绑定到模版显示

频道和文章列表联动实现（切换不同的频道id）

1. 为List组件定义 props 类型
2. 传递当前频道的 id，使用当前id获取列表



总结

1. 使用Vue + TS开发比较重要的是哪些类型的定义?

响应式数据 (ref) 的类型标注 - ref<类型>

axios返回数据 (res.data) 的类型标注 - axios.request<类型>

2. 类型的定义根据什么来做?

接口文档

3. 举一反三 (类型的思想)

凡是需要存储数据的地方都会涉及到类型标注

比如: 普通变量 / 对象属性



思考

思考一下如何把组件中除了Props之外的其他类型抽离到一个单独的data.d.ts文件中，然后在组件中导入类型使用，另外观察一下下面的类型定义如何使用泛型优化？

```
1 type ArticleResData = {  
2   data: {  
3     pre_timestamp: string  
4     results: ArticleItem[]  
5   }  
6   message: string  
7 }
```

```
1  
2 type ChannelRes = {  
3   data: {  
4     channels: ChannelItem[]  
5   }  
6   message: string  
7 }
```



传智教育旗下高端IT教育品牌