

Vue3 优雅的开发项目

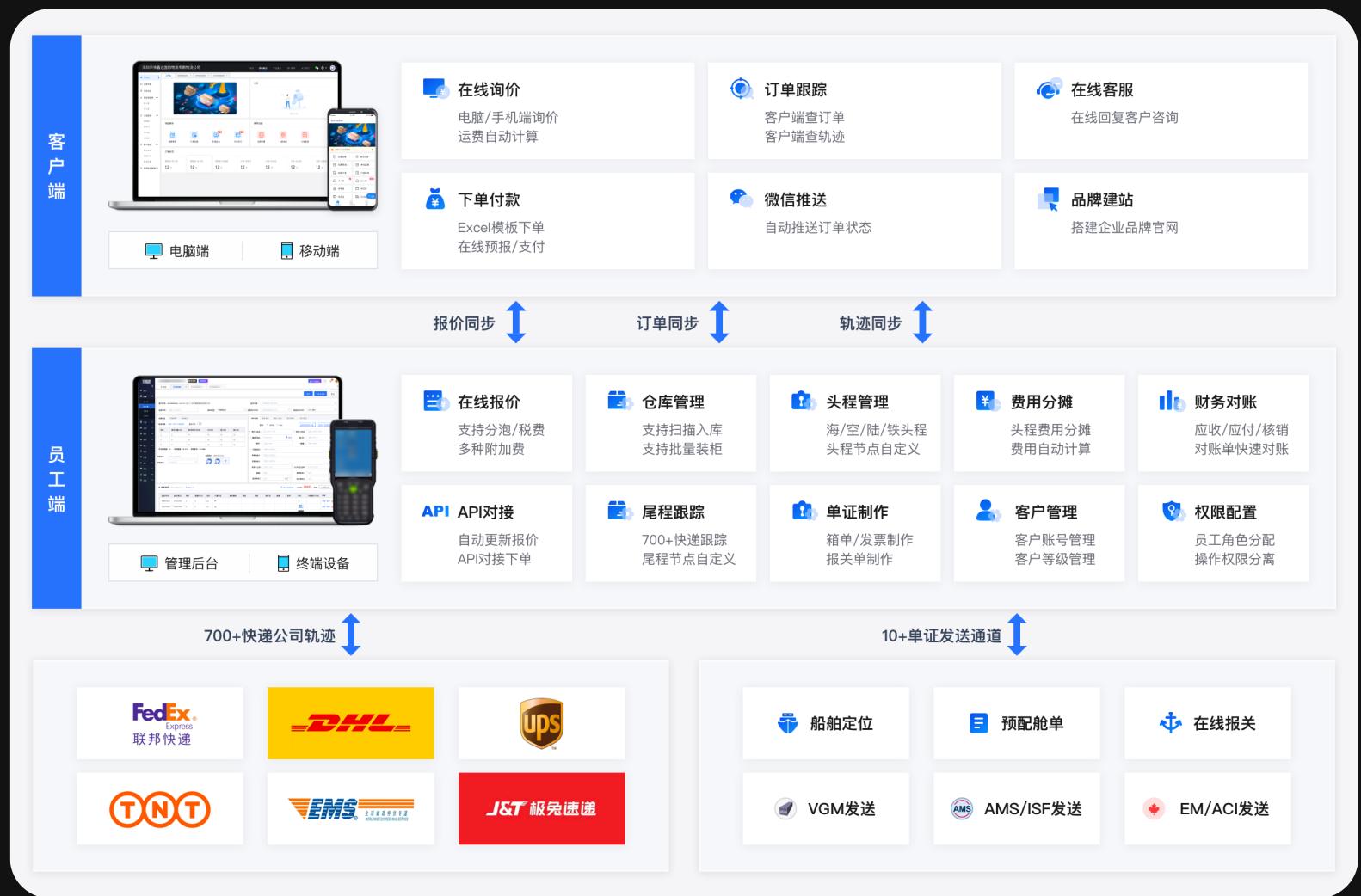
分享Vue3项目中好用的包及工具，提高开发效率，减少心智负担，提升开发体验，重拾开发乐趣

 跨境组 - TALKS

Suzhou, China 2022

跨境物流系统

一个基于vue3开发的多端项目
包括后台管理端、后台员工端
二级客户端 (PC、H5、小程序、公众号、App)



邵富旺·张超烨·王源·陈成·陶翔·陈彦彦·张辉?·?

talks.happyfly.top/2022/vue3-ecosystem

快速浏览

- 包管理器
- 工具链/脚手架
- 包的导入
- SVG图标
- Composition Api
- 全局数据状态管理
- CSS原子化引擎
- SSR/SSG
- 单元测试框架

想想要提前看看问了哪些问题，问题藏在了本次PPT源码里了哦，欢迎访问



Vue3用啥包管理器好呢？

npm or yarn ?

都可以，更推荐使用pnpm

Fast, disk space efficient package manager

- 磁盘空间利用非常高效

pnpm 内部使用基于内容寻址的文件系统来存储磁盘上所有的文件, 不会重复安装同一个包

- 包安装速度极快

因为下载包的数量与下载速度成正比, 包少了当然快

- 支持monorepo

用一个 git 仓库来管理多个子项目, 所有的子项目都存放在根目录的packages目录下, 那么一个子项目就代表一个package, 可替代lerna,
常用UI组件库如ElementPlus等都已使用了pnpm monorepo特性进行包管理。本次 Talk 源码即monorepo, 感兴趣的同学可以访问

- 安全性高

npm/yarn 的时候, 由于 node_module 的扁平结构, 可能会导致某些包的依赖关系不能被正确解析, 会出现这种非法访问的情况, 而pnpm 使用软链的方式将项目的直接依赖添加进模块文件夹的根目录, 避免了这种情况

不同的包管理器如何切换？

推荐安装 [antfu/ni](#) 包，可以识别当前项目所使用的的包管理器，并且可以自动切换到相应的包管理器，抹平不同的包管理器的命令差异，运行相应的脚本

```
npm i -g @antfu/ni
```

Nodejs版本如何切换？

macos 使用 [nvm](#)

windows 使用 [nvm-windows](#)

```
npm i -g nvm
```

vue3用啥工具链/脚手架？

能用vite就用vite, vue-cli4也可升级到vue-cli5或webpack5

vite

又快有稳，插件完善

- 极速的服务启动

- 使用原生 ESM 文件，使用 esbuild 预构建依赖， 使用Rollup 应用打包

- 轻量快速的热重载

- 无论应用程序大小如何，都始终极快的模块热重载（HMR）

- 配置简洁

- 对 TypeScript、JSX、CSS 等支持开箱即用

- 浏览器支持

- 可使用插件 [@vitejs/plugin-legacy](#) 兼容传统浏览器

vue-cli5/webpack5

相比v4，新增了一些特性，对ts支持更加友好

- webpack5内部实现持久化缓存

无需配置第三方插件，内部支持配置简单

- 更优的 tree-shaking

默认会进行tree-shaking, 增加了对一些 CommonJs 构造的支持，允许消除未使用的 CommonJs 导出，并从 require() 调用中跟踪引用的导出名称

- Module Federation 模块联邦

有点类似于云组件的概念，跟微前端的理念类似。它使 JavaScript 应用得以从另一个 JavaScript 应用中动态地加载代码 — 同时共享依赖。相当于 webpack 提供了线上 runtime 的环境，多个应用利用 CDN 共享组件或应用，不需要本地安装 npm 包再构建了

- 去除 Node.js Polyfills

需要开发者清楚项目需要引入哪些node polyfill进行配置，但是减少了代码的体积

如何优雅的导入包？

全量引入 or 按需引入？

更优的选择是自动导入

因为可以不再导入vue的API，也不用导入组件和注册组件，还能便捷看到这组件/API的类型推导

```
npm install -D unplugin-vue-components unplugin-auto-import
```

```
// vite.config.ts
import { defineConfig } from 'vite'
import AutoImport from 'unplugin-auto-import/vite'
import Components from 'unplugin-vue-components/vite'
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers'

export default defineConfig({
  plugins: [
    // 自动导入API Auto import APIs
    AutoImport({
      // 自动导入 Vue 相关函数, 如: ref, reactive, toRef 等
      imports: ['vue'],
      // ElementPlus组件解析器
      resolvers: [ElementPlusResolver()],
    }),
    // 自动注册组件 Auto import Components
    Components({
      resolvers: [ElementPlusResolver()],
    }),
  ],
})
```

自动生成TS类型文件

无需担心类型推导丢失

```
// Generated by 'unplugin-auto-import'  
declare global {  
    const computed: typeof import('vue')['computed']  
    const createApp: typeof import('vue')['createApp']  
    const customRef: typeof import('vue')['customRef']  
    ...  
}  
export {}
```

```
// generated by unplugin-vue-components  
import '@vue/runtime-core'  
  
declare module '@vue/runtime-core' {  
    export interface GlobalComponents {  
        BaseButton: typeof import('../components/BaseButton.vue')['default']  
        ...  
        ElButton: typeof import('element-plus/es')['ElButton']  
    }  
}
```

如何优雅的使用svg组件和icon图标？

iconfont or svglcon ?

看看组件库是怎么设计icon的

通用做法是将icon组件化，如下ElementPlusIcon全局导入

```
import * as ElementPlusIconsVue from '@element-plus/icons-vue'

const app = createApp(App)
for (const [key, component] of Object.entries(ElementPlusIconsVue)) {
  app.component(key, component)
}
```

```
<el-icon :size="size" :color="color">
  <Edit />
</el-icon>
```

我们似乎可以很方便的配合el-icon使用图标组件，那本地图标咋办?🤔

本地的svg图标集应该如何配合使用

vite可以借助vite-svg-loader引入svg组件，webpack5也可以配置assetModule引入svg组件

```
// vite 配置 vite-svg-loader
import foo from "@/assets/foo.svg?component"
```

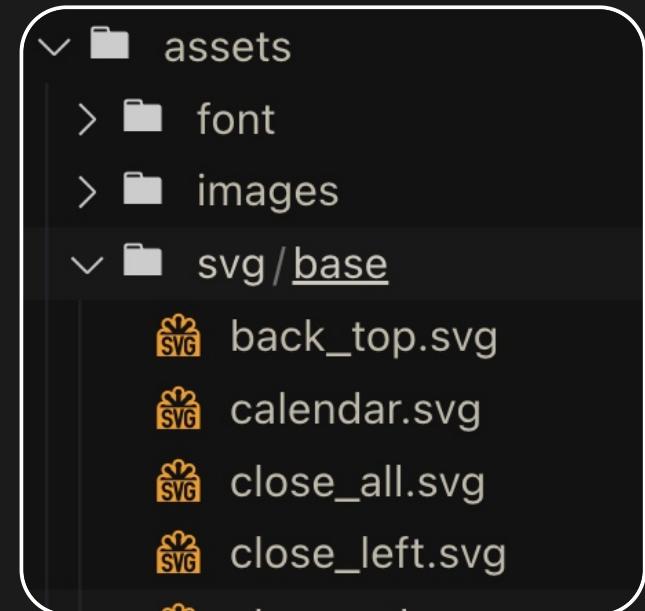
```
// webpack5 配置 assetModule
import foo from "@/assets/svg/foo.svg";
```

使用起来没什么大问题，但是不够优雅👻

自动导入配合icones图标集

```
npm i -D unplugin-icons
```

```
import Icons from 'unplugin-icons/vite'
import { FileSystemIconLoader, IconsResolver } from 'unplugin-icons/loaders'
export default defineConfig({
  plugins: [
    // 自动导入图标组件
    AutoImport({
      resolvers: [
        IconsResolver({
          prefix: 'Icon',
        })
      ],
    }),
    // 标识自定义图标集
    Components({
      resolvers: [
        IconsResolver({
          enabledCollections: ['ep'],
          customCollections: ['base'],
        })
      ],
    }),
  ],
})
```



```
Icons({
  compiler: 'vue3',
  autoInstall: true, // 自动安装
  //custom图标集 给svg文件设置fill="currentColor"属性，使图标的颜色具有适应性
  customCollections: [
    base: FileSystemIconLoader('src/assets/svg/base', svg =>
      svg.replace(/^<svg /, '<svg fill="currentColor" '),
    ),
  ],
})
```

```
<!-- elementplus 图标集 -->
<el-icon :size="size" :color="color">
  <i-ep-arrow-down />
</el-icon>

<!-- 本地自定义base图标集 -->
<el-icon :size="size" :color="color">
  <i-base-back-top />
</el-icon>
```

至于ICONFONT，当然我们也可以将其封装成组件使用，可参考此DEMO

```
<script lang="ts" setup>
import { useAttrs, computed } from 'vue'
const { icon } = defineProps<{icon: string}>()
const attrs = useAttrs()
const isUni = computed(() => Object.keys(attrs).includes('uni') || attrs?.iconType === 'uni')
const isSvg = computed(() => Object.keys(attrs).includes('svg') || attrs?.iconType === 'svg')
</script>
```

```
<template>
<i v-if="isUni" class='iconfont' v-bind="$attrs">{{ icon }}</i>
<svg v-else-if="isSvg" class="icon-svg" aria-hidden>
  <use :xlink:href="#${icon}"></use>
</svg>
<i v-else class="iconfont" v-bind="$attrs" :class="icon"></i>
</template>
```

我是 ↗ 的图标 囧 囧

```
<Iconfont uni icon="#xe9a2;" />
<Iconfont icon="iconxiadan" />
```

如何优雅的使用Composition API?

看看vueuse是如何设计的

VueUse

VueUse是一个基于Composition API的实用函数集合

- 同时兼容 Vue 2 和 Vue 3
- Tree-shakeable ESM
- TypeScript
- CDN 兼容
- 核心包含 110+ 组合式函数
- 丰富的生态系统 8+ 扩展包

可组合的函数

可复用逻辑的集合，每一个函数都可以独立使用，专注点分离

```
export function useDark(options: UseDarkOptions = {}) {
  const preferredDark = usePreferredDark() // <--
  const store = useLocalStorage('vueuse-dark', 'auto') // <--

  return computed<boolean>({
    get() {
      return store.value === 'auto'
        ? preferredDark.value
        : store.value === 'dark'
    },
    set(v) {
      store.value = v === preferredDark.value
        ? 'auto' : v ? 'dark' : 'light'
    },
  })
}
```



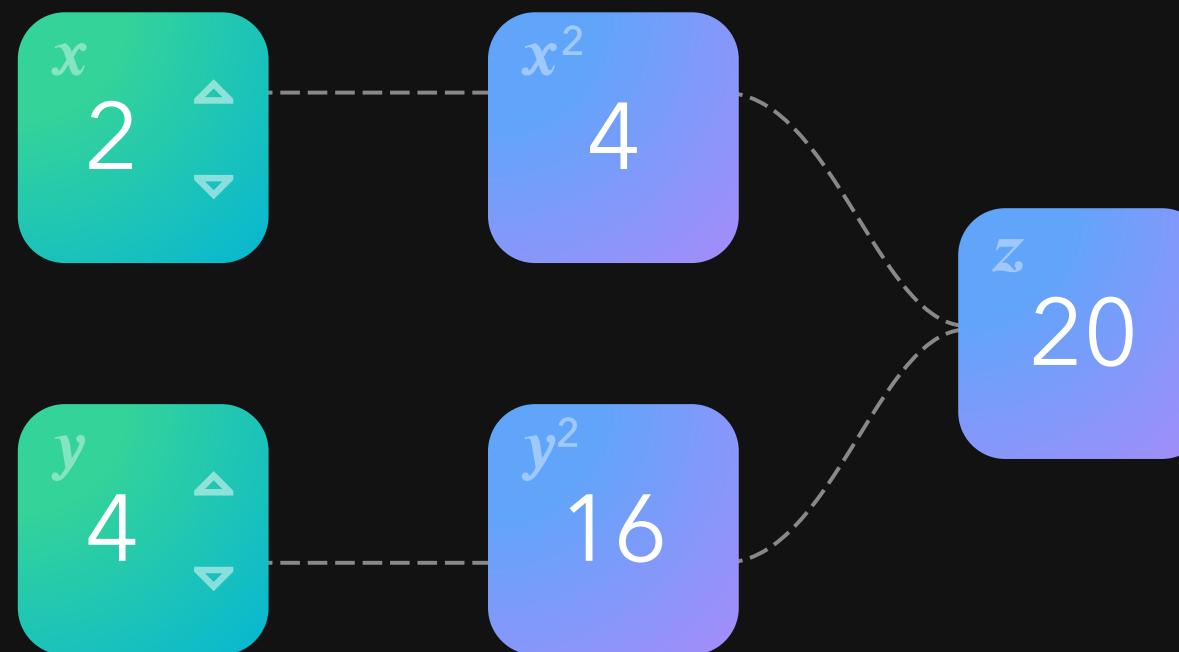
在 VueUse 中可用: usePreferredDark useLocalStorage useDark

建立"连结"

不同于 React, Vue 的 `setup()` 只会在组件建立时执行一次，并建立数据与逻辑之间的连结

- 建立 输入 → 输出 的连结
- 输出会自动根据输入的改变而改变

$$z = x^2 + y^2 = 2 \times 2 + 4 \times 4 = 20$$



Excel 中的公式

A screenshot of an Excel spreadsheet titled "SUM". The formula bar shows the formula $=C2/B2$. The spreadsheet has four columns: Student, Total Marks, Achieved Marks, and Percentage. The Percentage column contains the formula $=C2/B2$, which is highlighted with a red border. The data rows show student names Ramu, Rajitha, Komala, Patil, Pursi, and Gayathri, along with their total marks and achieved marks.

	A	B	C	D	E
1	Student	Total Marks	Achieved Marks	Percentage	
2	Ramu	600	490	=C2/B2	
3	Rajitha	600	483		
4	Komala	600	448		
5	Patil	600	530		
6	Pursi	600	542		
7	Gayathri	600	578		
8					

如何优雅的进行状态管理？

vuex4 ?

对于全局的数据状态管理，按照VueUse结合Composition API设计思路，composables函数属性共享，已然支持状态管理，但是像vuex一样，我们需要封装一些好用的API才能适用于大的项目。

无需担心，Pinia 已经帮我们做了这件事情



The Vue Store that you will enjoy using

可以认为是vuex4的升级版本

- API 简洁，对Vue3支持更好
- TypeScript 可靠的类型推断支持
- 支持模块化
- 支持插件化，可实现事务，同步本地存储

可以安装以下插件支持持久化 `npm i pinia-plugin-persistedstate`

- Vue devtools 插件支持
- 体积小，约1kb

TodoList

Please input...

Add

UnFinished

Finished

[]

```
import { defineStore } from 'pinia'
interface Todo {
  text: string, id: number, isFinished: boolean
}
export const useTodos = defineStore('todos', {
  // 开启数据持久化
  persist: true,
  state: () => ({
    todos: [] as Todo[],
    /** @type {'all' | 'finished' | 'unfinished'} */
    filter: 'all',
    // 将自动推断出ts类型为number
    nextId: 0,
  }),
  getters: {
    finishedTodos(state) {
      return state.todos.filter((todo) => todo.isFinished)
    },
    unfinishedTodos(state) {
      return state.todos.filter((todo) => !todo.isFinished)
    },
  }
})
```

```
/**
 * @returns {{ text: string, id: number, isFinished: boolean }[]}
 */
filteredTodos(state) {
  if (this.filter === 'finished') {
    // 调用其他的getters计算返回新值 ✨
    return this.finishedTodos
  } else if (this.filter === 'unfinished') {
    return this.unfinishedTodos
  }
  return this.todos
},
},
actions: {
  // 接受一些参数， 返回一个promise 或者 void
  addTodo(text) {
    // 你可以直接修改state
    this.todos.push({ text, id: this.nextId++, isFinished: false })
  },
  deleteTodo(item) {
    const index = this.todos.indexOf(item)
    this.todos.splice(index, 1)
  },
},
})
}
```

我想要在项目中使用CSS原子化引擎

tailwinds or windicss ?

vite 构建推荐 UnoCSS / 其他构建推荐 Windicss

性能强且极具灵活性的即时原子化 CSS 引擎, 可以认为是 Windi CSS 团队的一个激进的实验, 现在版本已稳定

■ 引擎

{ UnoCSS 是一个引擎, 而非一款框架, 所有功能可以通过预设和内联配置提供 }

■ 预设

{ 默认的 @unocss/preset-uno 预设是一系列流行的原子化框架的 通用超集, 包括了 Tailwind CSS, Windi CSS, Bootstrap, Tachyons 等 }

■ 灵活性

```
<button class="bg-blue-400 hover:bg-blue-500 text-sm text-white font-mono font-light py-2 px-4 rounded border-2 border-
```

```
<button  
  bg="blue-400 hover:blue-500 dark:blue-500 dark:hover:blue-600"  
  text="sm white"  
  font="mono light"  
  p="y-2 x-4"  
  border="2 rounded blue-200"  
>  
  Button  
</button>
```

UnoCSS的性能表现

UnoCSS 可以比 Tailwind 的 JIT 引擎快 200 倍

- 跳过解析，不使用 AST

从内部实现上看，Tailwind 依赖于 PostCSS 的 AST 进行修改，而 Windi 则是编写了一个自定义解析器和 AST。考虑到在开发过程中，这些工具 CSS 的并不经常变化，UnoCSS 通过非常高效的字符串拼接来直接生成对应的 CSS 而非引入整个编译过程。同时，UnoCSS 对类名和生成的 CSS 字符串进行了缓存，当再次遇到相同的实用工具类时，它可以绕过整个匹配和生成的过程

- 单次迭代

Windi CSS 和 Tailwind JIT 都依赖于对文件系统的预扫描，并使用文件系统监听器来实现 HMR。文件 I/O 不可避免地会引入开销，而你的构建工具实际上需要再次加载它们 而 UnoCSS 直接利用已经被工具读取过的内容

我的项目要求SSR/SSG

nuxt3不二之选

Nuxt3

Nuxt 3 的重构精简了内核，并且让速度更快，开发体验更好

- 开发更快

- 用动态服务端代码分割来优化冷启动，由 nitro 引擎提供能力

- 更轻量

- 以现代浏览器为目标的情况下，服务器部署和客户端产物最多可达 75 倍的减小

- Hybrid

- 增量静态生成和其他高级模式现在都成为可能

- Suspense

- 导航前后皆从任何组件中获取数据

- Composition API

- 使用 Composition API 和 Nuxt 3 的 composables 实现真正的可复用性

- Nuxt CLI

更多的信息和快速修复，在浏览器中高效工作

- Nuxt Kit

全新的基于 TypeScript 和跨版本兼容的模块开发

- Webpack 5

更快的构建速度和更小的构建产物，并且零配置

- Vite

用 Vite 作为你的打包器，体验轻量级的快速 HMR

- Vue3

Vue3会成为您下一个应用的坚实基础。

- TypeScript

由原生 TypeScript 和 ESM 构成，没有额外的步骤

Nuxt3 服务器引擎“Nitor”

- 跨平台支持Node.js、浏览器、service-worker等
- 支持Serverless 开箱即用
- 约定式路由
- 自动代码分割，异步加载模块
- 静态 + 服务器站点的混合模式
- 热重载的开发服务器

Nuxt3 app demo

Nuxt3 app demo 源码

如果只是想要写个SSG静态站点文档，那么还可以选择 Vitepress (VitePress is VuePress' little brother, built on top of Vite.)

我写了还不错的包，用什么测试好呢？

Jest?

我要更快，那么请选择Vitest

Vitest 是一个由 Vite 提供支持的极速单元测试框架

- 与 Vite 通用的配置、转换器、解析器和插件
- 使用你的应用程序中的相同配置来进行测试
- 智能文件监听模式，就像是测试的 HMR
- 支持测试 Vue、React、Lit 等框架中的组件
- 开箱即用的 TypeScript / JSX 支持
- ESM 优先，支持模块顶级 await
- 通过tinypool使用Worker线程尽可能多地并发运行
- 套件和测试的过滤、超时、并发配置
- Jest 的快照功能
- 内置 Chai 进行断言 + 与 Jest expect 语法兼容的 API
- 内置用于对象模拟(Mock)的 Tinyspy
- 使用 jsdom 或 happy-dom 用于 DOM 模拟
- 通过 c8 来输出代码测试覆盖率
- 类似于 Rust 语言的 源码内联测试

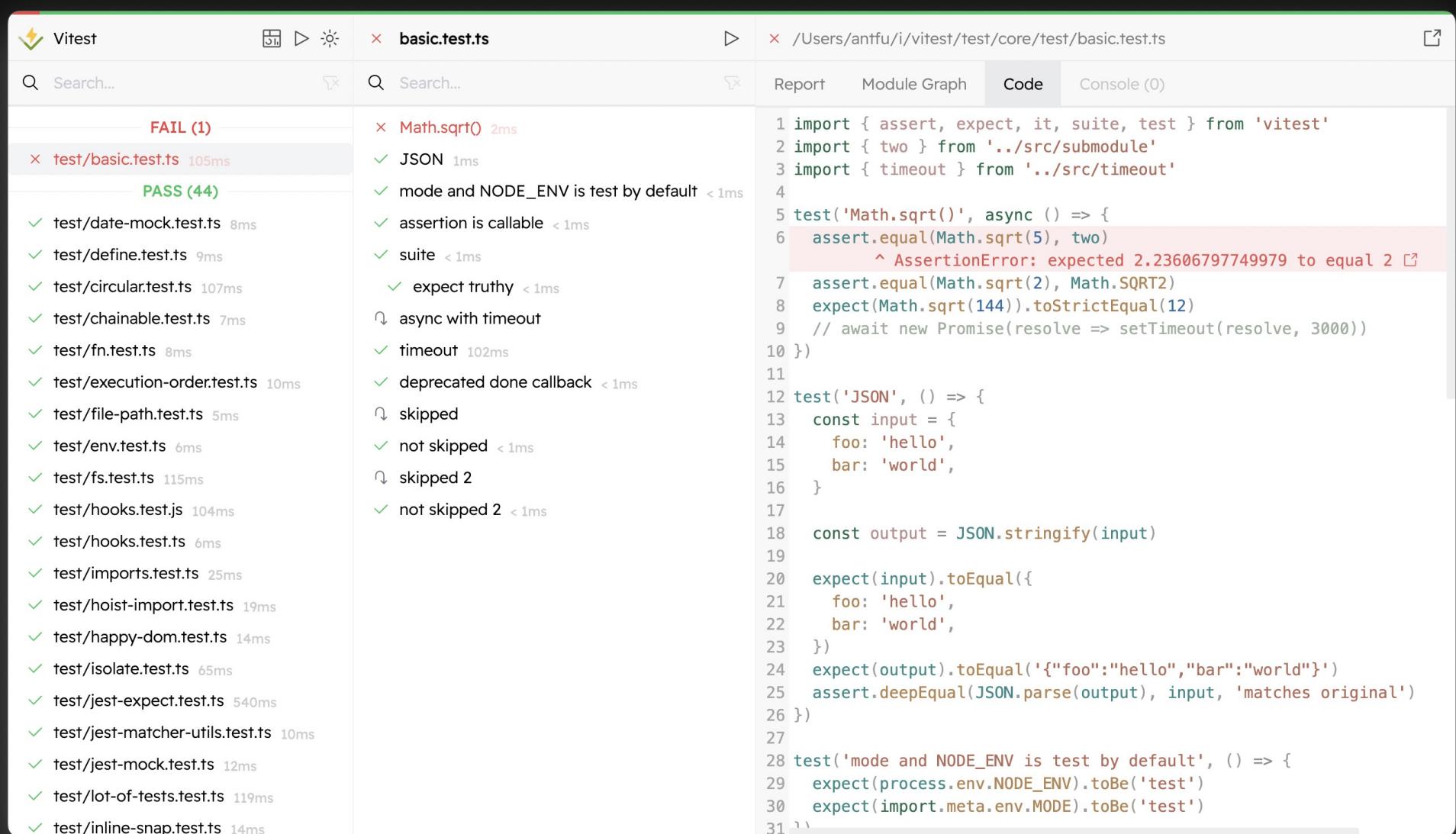
Vitest Demo

对比Jest框架

vite-jest提供了一套跟 Vite 集成的 Jest，所以 Jest 也是 Vite 生态下单元测试框架的可选项。但是vitest还支持非vite运行的项目。简单来说，Vitest 与大多数 Jest API 和生态系统库都有较好的兼容性，可以无缝的将 Jest 替换成 Vitest，但由于vitest的即时热重载，更快的运行速度，开发体验会有飞跃的提升

Vitest Ui

Vitest在运行测试时有一个开发服务器，因此Vitest可以提供如下这样的 UI 界面来查看并与测试交互



The screenshot shows the Vitest UI interface. On the left, there's a sidebar with a tree view of test files and a search bar. The main area has two tabs: 'Report' and 'Code'. The 'Report' tab is active, showing a summary of test results: FAIL (1) and PASS (44). It lists individual test cases with their status, file path, and execution time. The 'Code' tab is also visible. On the right, there's a code editor window displaying the source code for a file named 'basic.test.ts'. The code uses Vitest's API to run various assertions, including a failing assertion where it expects 2.23606797749979 to equal 2.

```
1 import { assert, expect, it, suite, test } from 'vitest'
2 import { two } from '../src/submodule'
3 import { timeout } from '../src/timeout'
4
5 test('Math.sqrt()', async () => {
6   assert.equal(Math.sqrt(5), two)
7   ^ AssertionError: expected 2.23606797749979 to equal 2
8   assert.equal(Math.sqrt(2), Math.SQRT2)
9   expect(Math.sqrt(144)).toStrictEqual(12)
10  // await new Promise(resolve => setTimeout(resolve, 3000))
11
12 test('JSON', () => {
13   const input = {
14     foo: 'hello',
15     bar: 'world',
16   }
17
18   const output = JSON.stringify(input)
19
20   expect(input).toEqual({
21     foo: 'hello',
22     bar: 'world',
23   })
24   expect(output).toEqual('{"foo":"hello","bar":"world"}')
25   assert.deepEqual(JSON.parse(output), input, 'matches original')
26 })
27
28 test('mode and NODE_ENV is test by default', () => {
29   expect(process.env.NODE_ENV).toBe('test')
30   expect(import.meta.env.MODE).toBe('test')
31 })
```

你以上说的我大多接受，但我并不想配置
有现成的项目模板？

项目模板&插件&文章

App

- [vitesse](#)
- [vitesse-nuxt3](#)

Vscode

- [volar](#)
- [iconify](#)
- [i18n-ally](#)
- [unocss](#)
- [windicss-intellisense](#)

Admin

- [vue-pure-admin](#)

Read

- [关于现代包管理器的深度思考](#)
- [重新构想原子化 CSS](#)
- [vue函数的最佳实践与技巧](#)
- [Vue.js设计与实现](#)

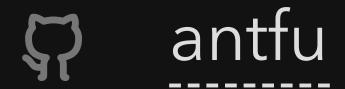
谢谢！再见！

彩蛋！ 感谢大佬开源！

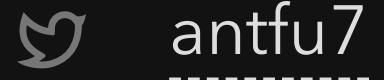
Anthony Fu

Vue·Vite·Nuxt 核心团队成员

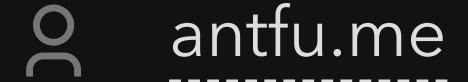
VueUse, Slidev, Type Challenges 等项目创作者



[antfu](#)



[antfu7](#)



[antfu.me](#)



[Discord](#)



[Anthony Fu](#)

