

产险科技中心  
**前端培优<sup>+</sup>**  
**训练营**

2023年

# 寻找并实现Vue2&3的核心区别

主讲人：宸羽老师

时间：2023年12月13日

# 编辑器

推荐的 IDE 配置是 [Visual Studio Code](#) + [Volar 扩展](#)

# 多个版本的 Nodejs 安装

- nvm

node version management

node 版本: v16.13.0   v20.10.0

- <https://github.com/coreybutler/nvm-windows/releases>

# devtool

- <https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd>
- <https://github.com/vuejs/devtools>

# CSS

## 原子样式

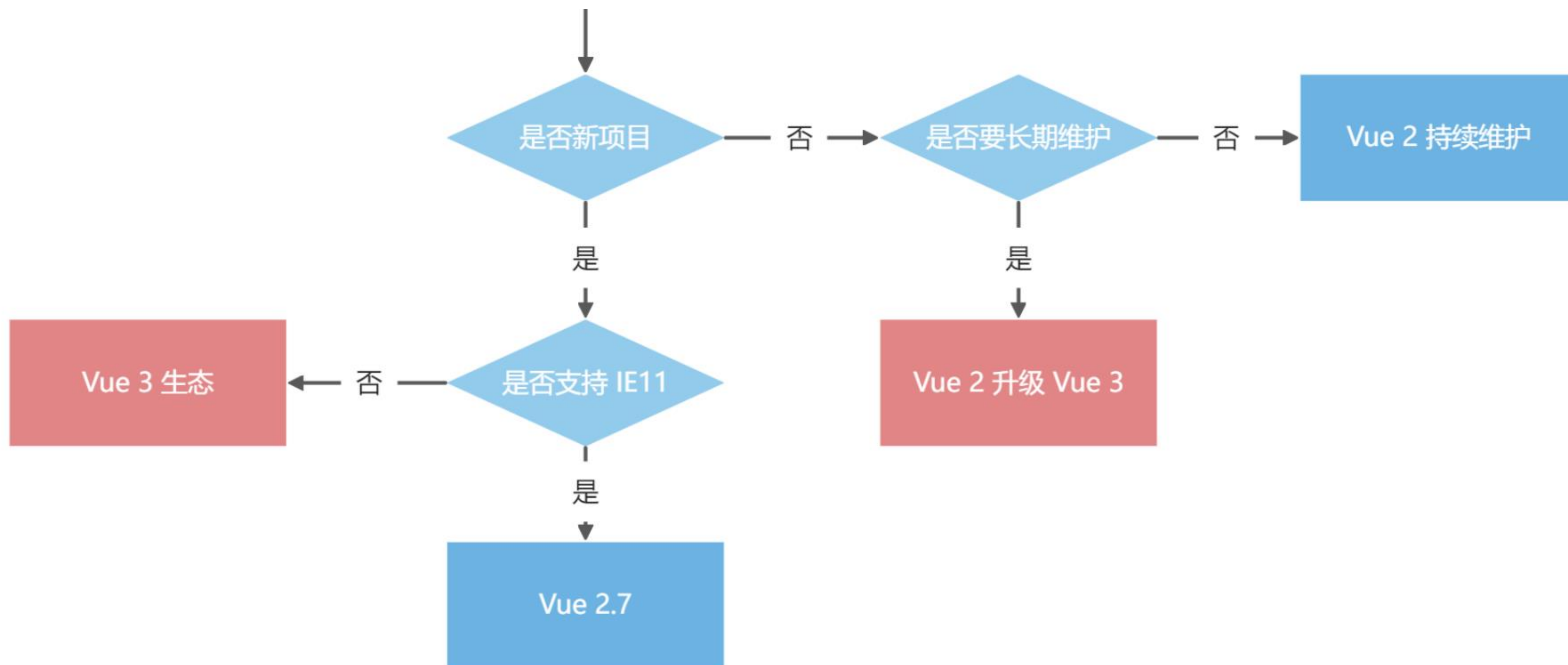
- <https://tailwindcss.com/>

# vue3 在线编码

- <https://play.vuejs.org/#eNp9kc1uwyAQhF8FoR4SJUpU5RY5Vn+UQ3toq7ZHLineuCQYECyuJcvv3gUraQ5VfDIzH8Ms9PzeuUUbgga+5MAVC4/QOoRRGYFGptux7aaPBYSiWaZn1r4hoDbuTWsnjRvBdVQle5hVroFiOPrHF8jLQFEF65ZAFwOhI0YAsp7MNu/Gwn9xOE7aPRqKiAyh3MmV9OjNjs5kwQ0odc0o+5xikNXtVLw7BGpqBYMYEI7ZxSoN/dSkoCL5OMYw+aqu1/XnOGvol85Muv0Ee/9EPoUua4G8eAvgWBD97uPM14GhvP16go/+z2dgqaqKvmO8Qrl6p44g9RFNR7Qsut31qnPWotP0Zth2CCaehUtFEDpkXnB7y8crof3VXi1XeRxfKh1+P8K4b>



# 到底用2还是3



# vue3 代码

main.js

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'
import './index.css'

createApp(App).use(router).use(store).mount('#app')
```



# vue 3 对比

## Vue 2

```
import Vue from 'vue';  
import App from './App.vue';  
import router from './router';  
import store from './store';
```

```
Vue.config.productionTip = false;
```

```
new Vue({  
  router,  
  store,  
  render: h => h(App)  
}).$mount('#app');
```

## Vue 3

```
import { createApp } from 'vue';  
import App from './App.vue';  
import router from './router';  
import store from './store';
```

```
createApp(App)  
  .use(router)  
  .use(store)  
  .mount('#app');
```



# createApp

- 新增了 App 的概念。全局的组件、插件都独立地注册在这个 App 内部
- createApp 还移除了很多我们常见的写法，比如在 createApp 中，就不再支持 filter、\$on、\$off、\$set、\$delete 等 API。



## Vue 2

```
import Vue from 'vue';  
import Vuex from 'vuex';  
  
Vue.use(Vuex);  
  
export default new Vuex.Store({  
  state: {},  
  mutations: {},  
  actions: {},  
  modules: {}  
});
```

## Vue 3

```
import { createStore } from 'vuex';  
  
export default createStore({  
  state: {},  
  mutations: {},  
  actions: {},  
  modules: {}  
});
```



```
import Vue from 'vue';
import VueRouter from 'vue-router';
import Home from '../views/Home.vue';

Vue.use(VueRouter);

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  }
];

export default new VueRouter({
  routes
});
```

→

```
import {
  createRouter,
  createWebHashHistory
} from 'vue-router';
import Home from '../views/Home.vue';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  }
];

export default createRouter({
  history: createWebHashHistory(),
  routes
});
```



@vue/compat

# Composition API + <script setup>

在 Composition API 中，所有的功能都通过全局引入的方式使用的

并且通过 <script setup>，定义的变量、函数和引入的组件，都不需要额外的生命周期，就可以直接在模板中使用。



# CSS

```
<style lang="scss" scoped>
@mixin tip-line {
  &::before {
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    width: v-bind(tipLineWidth);
    height: 100%;
    background-color: var(--v3-side
  }
}
```

# ref 获取

```
<template>
  <div id="app">
    <div ref="hello">1213平安你好</div>
  </div>
</template>

<script>
export default {
  mounted () {
    // <div>1213平安你好</div>
    console.log(this.$refs.hello)
  }
}
</script>
```

# 没有 this

- Vue3 中通过 ref 访问元素节点与 Vue2 不太一样，在 Vue3 中我们是没有 this 的，所以当然也没有 this.\$refs。想要获取 ref，我们只能通过声明变量的方式。

```
import { ref, getCurrentInstance } from "vue";  
export default {  
  props: { ...  
  },  
  setup() {  
    // @ts-ignore  
    const { ctx } = getCurrentInstance();
```

# ref 和 reactive

都是用来定义响应式数据

```
import {ref} from 'vue'
let num = ref(1)
function add(){
  num.value++
}
```

# ref 简单设置

---

```
<script setup>
import { ref } from 'vue'

const isShanghai = ref(true)
</script>

<template>
  <button @click="isShanghai = !isShanghai">确认是否 {{ isShanghai }}</button>

  <h1 v-if="isShanghai">在上海进行 2 天课程学习</h1>
  <h1 v-else>不是今天哦</h1>
</template>
```



# ref 设置数字

ref 定义了一个数字

```
<script setup>
import { ref } from 'vue'

let score = ref(2.5);
function random(min, max) {
  // 首先, 计算范围内的数字数量
  var numCount = max - min + 1;
  // 然后, 生成一个随机数, 它在0到numCount之间
  var randomNum = Math.floor(Math.random() * numCount);
  // 最后, 将随机数与最小值相加, 得到最终结果
  var result = randomNum + min;
  // 返回结果
  return result;
}

function updateScore() {
  var num = random(1, 21)
  console.log('num', num)
  score.value = num;
}
</script>

<template>
  <h1>你的评分是 {{ score }}</h1>
  <button @click="updateScore">随机修改评分</button>
</template>
```

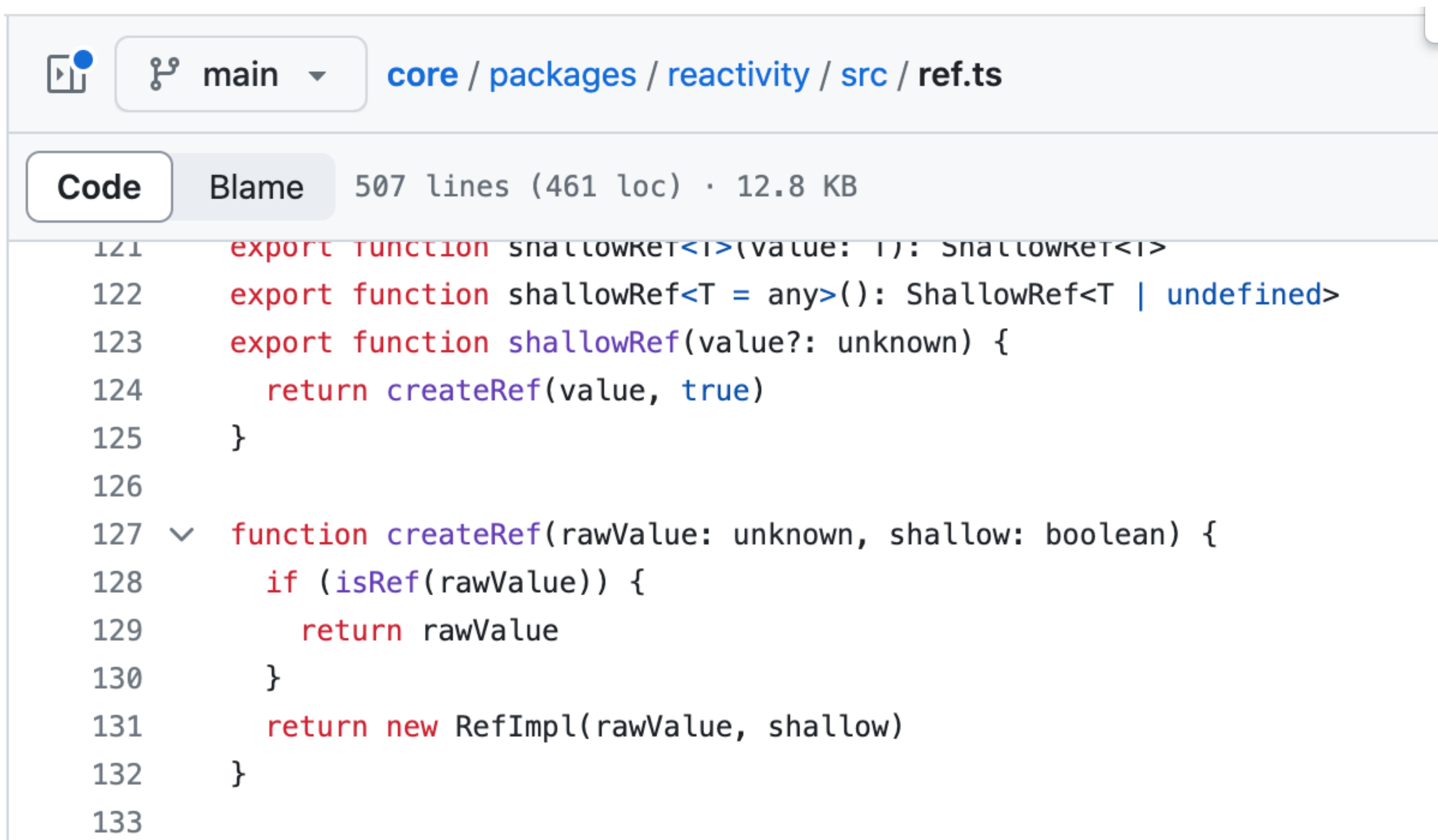
reactive:

- \*、响应式是更深层次的，底层是将传入的数据包装成一个proxy;
  - \*、推荐定义复杂的数据类型； 参数必须是对象或者数组，
- 如果想要使用reactive让某个数值变成响应式，要在外层包裹一个对象{};
- 如果要想对象的某个元素实现响应式时比较麻烦。需要使用 toRefs

ref:

- \*、函数参数可以是基本数据类型，也接受对象类型;
- \*、如果参数是对象类型时，其实底层的本质还是reactive, 系统会自动根据我们给ref传入的值转换;
- \*、在template中访问，系统会自动添加.value;在js中需要手动.value;

# ref 的深层



The screenshot shows a code editor interface for a file named `ref.ts` located at `core / packages / reactivity / src /`. The file is 507 lines long (461 loc) and 12.8 KB. The code defines several functions related to reactive references:

```
121 export function shallowRef<T>(value: T): ShallowRef<T>
122 export function shallowRef<T = any>(): ShallowRef<T | undefined>
123 export function shallowRef(value?: unknown) {
124     return createRef(value, true)
125 }
126
127 function createRef(rawValue: unknown, shallow: boolean) {
128     if (isRef(rawValue)) {
129         return rawValue
130     }
131     return new RefImpl(rawValue, shallow)
132 }
133
```

3.2 版本之后都用  
ref?

# computed

使用**计算属性**来描述依赖响应式状态的复杂逻辑

```
import {  
  computed  
} from 'vue'  
  
const isOnlineSvg = computed()  
=> /^https?:/.test(props.icon))
```

computed() 方法期望接收一个 getter 函数，返回值为一个计算属性 ref。

# 生命周期

变化不大，只是名字大部分需要 + on，功能上类似

# 父子生命周期

父setup

父onBeforeMount

子setup

子onBeforeMount

子onMounted

父onMounted

修改父组件数据，触发父组件 onBeforeUpdate

修改子组件数据，触发子组件 onBeforeUpdate



# template 变化

组件现在支持有多个根节点

```
<template>  
  <header>...</header>  
  <main v-bind="$attrs">...</main>  
  <footer>...</footer>  
</template>
```

# useMouse

声明周期

ref

```
import { ref, onMounted, onUnmounted } from 'vue'
function useMouse() {
  const x = ref(0)
  const y = ref(0)
  const update = e => { ...
  }
  onMounted(() => {
    window.addEventListener('mousemove', update)
  })
  onUnmounted(() => {
    window.removeEventListener('mousemove', update)
  })
  return { x, y }
}
export default useMouse;
```

# defineProps

```
import { defineProps, defineEmits, computed, ref } from 'vue';  
let props = defineProps({  
  modelValue: Number,  
  theme: { type: String, default: 'orange' }  
})
```

# defineProps

```
<script setup>
const props = defineProps(['name'])

console.log(props.name)
</script>
```

```
export default {
  props: ['name'],
  setup(props) {
    // setup() 接收 props 作为第一个参数
    console.log(props.name)
  }
}
```

传递给 `defineProps()` 的参数和提供给 `props` 选项的值是相同的，两种声明方式背后其实使用的都是 `prop` 选项

# defineProps 宏命令

- defineProps 仅在 `<script setup>` 中可用的编译宏命令，不需要导入。
- 声明的 props 会自动暴露给模板。
- defineProps 会返回一个对象，包含了可以传递给组件的所有 props

# 父子组件 props

```
<template>
  <BlogPost
    v-for="post in posts"
    :key="post.id"
    :name="post.name"
  ></BlogPost>
</template>

<script setup>
import { ref } from 'vue'
import BlogPost from './BlogPost.vue'

const posts = ref([
  { id: 1, name: '上海学习' },
  { id: 2, name: '北京学习' },
  { id: 3, name: '深圳学习' }
])
</script>
```

```
<script setup>
const props = defineProps(['name'])
console.log('props', props)
</script>
```

```
<template>
  <p>{{ name }}</p>
</template>
```

## Proxy(Object)

- ▶ `[[Handler]]`: ReadonlyReactiveHandler
- ▶ `[[Target]]`: Proxy(Object)
- `[[IsRevoked]]`: false



编译后

- `const __module__ = __modules__["src/BlogPost.vue"] = { [Symbol.toStringTag]: "Module" }`
- `import { toDisplayString as _toDisplayString, openBlock as _openBlock, createElementBlock as _createElementBlock } from "vue"`
- 
- `const __sfc__ = {`
- `__name: 'BlogPost',`
- `props: ['name'],`
- `setup(__props) {`
- `const props = __props`
- `console.log('props', props)`
- `return (_ctx, _cache) => {`
- `return (_openBlock(), _createElementBlock("p", null, _toDisplayString(__props.name), 1 /* TEXT */))`
- `}`
- `}`
- 
- `}`
- `__sfc__.__file = "src/BlogPost.vue"`
- `__module__.default = __sfc__`

# defineProps


- 所有的 props 都遵循着单向绑定原则，props 因父组件的更新而变化，自然地将新的状态向下流往子组件，而不会逆向传递。

# defineEmits

```
import { defineProps, defineEmits, computed, ref } from 'vue';  
  
let emits = defineEmits(['update:modelValue'])
```

# 综合案例实战

信息：  
地址：中国平安  
总金额：33

 地址：中国平安

短期意外险	单价: 5	-	1	+
境外留学险	单价: 8	-	1	+
意外伤害险	单价: 10	-	2	+

# App.vue

```
<template>
  <div class="app">
    <info :text="state.text" :list="state.list" />
    <text :text="state.text" @updateText="updateText" />
    <list
      :list="state.list"
      @increase="increase"
      @decrease="decrease"
    />
  </div>
</template>
```

# App.vue

```
const updateText = (text) => {
  state.text = text;
}
const increase = (index) => {
  state.list[index].count += 1;
}
const decrease = (index) => {
  if (state.list[index].count > 0) {
    state.list[index].count -= 1;
  }
}
```

```
<script setup >
import Info from './components/Info.vue';
import Text from './components/Text.vue';
import List from './components/List.vue';

import { reactive } from 'vue';
const state = reactive({
  text: '苏州市高新区科技城',
  list: [
    { name: '橘子', price: 5, count: 0 },
    { name: '橙子', price: 8, count: 0 },
    { name: '香蕉', price: 10, count: 0 },
  ]
});
```

# Info.vue

```
<template>
  <div class="v-info">
    <div>信息: </div>
    <div>地址: {{props.text}}</div>
    <div>总金额: <span>{{totalPrice}}</span></div>
  </div>
</template>
```

```
<script setup>
  import { ref, watch } from 'vue';
  const props = defineProps({
    text: String,
    list: Array,
  });
  const totalPrice = ref(0);
  watch(props, () => {
    const list = props.list;
    let total = 0;
    list.forEach((item) => {
      total += item.price * item.count;
    });
    totalPrice.value = total;
  })
</script>
```



# List.vue

```
<template>
  <div class="v-list">
    <div class="v-list-item"
      :key="index"
      v-for="(item, index) in list">
      <span class="text">{{item.name}}</span>
      <span class="text">单价: {{item.price}}</span>
      <button class="btn" @click="onClickDecrease(index)">-</button>
      <span class="count"> {{item.count}}</span>
      <button class="btn" @click="onClickIncrease(index)">+</button>
    </div>
  </div>
</template>
```



# List.vue

```
<script setup>
const props = defineProps({
  list: Array,
})
const emits = defineEmits(['increase', 'decrease'])
const onClickIncrease = (index) => {
  emits('increase', index)
}
const onClickDecrease = (index) => {
  emits('decrease', index)
}
</script>
```

# Text.vue

```
<template>
  <div class="v-text">
    <span>地址: </span>
    <input :value="props.text" @input="onInput" />
  </div>
</template>
```

```
<script setup >
const props = defineProps({
  text: String,
});
const emits = defineEmits(['updateText']);
const onInput = (e) => {
  emits('updateText', e.target.value);
}
</script>
```

# computed

```
const themeObj = {
  'black': '#00',
  'white': '#fff',
  'red': '#f5222d',
  'orange': '#fa541c',
  'yellow': '#fadb14',
  'green': '#73d13d',
  'blue': '#40a9ff',
}

const fontstyle = computed(() => {
  return `color:${themeObj[props.theme]}`;
})
```

# 构建工具

- webpack
- vite
- parcel
- esbuild
- rollup
- grunt
- gulp

# vite

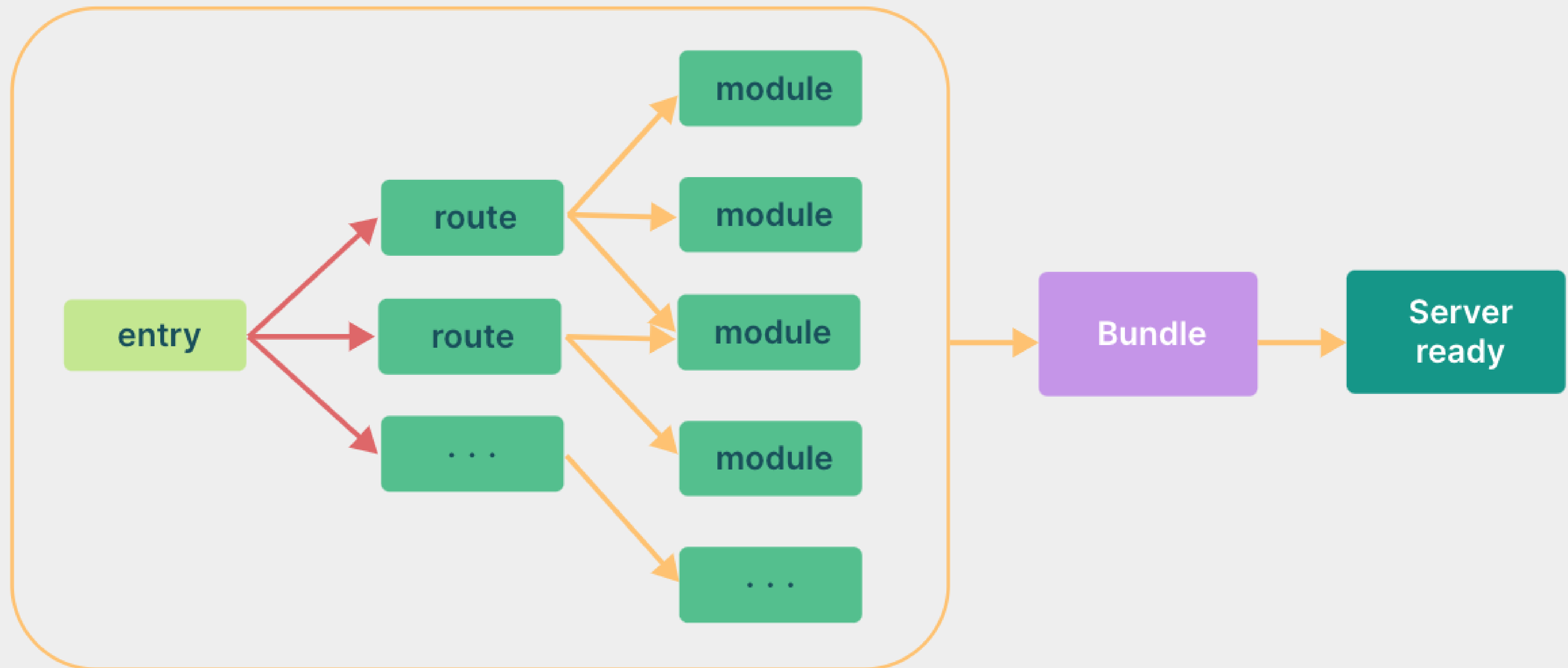
- Vite 使用 **esbuild** 预构建依赖。

esbuild 使用 Go 编写，并且比以 JavaScript 编写的打包器预构建依赖快 10-100 倍。

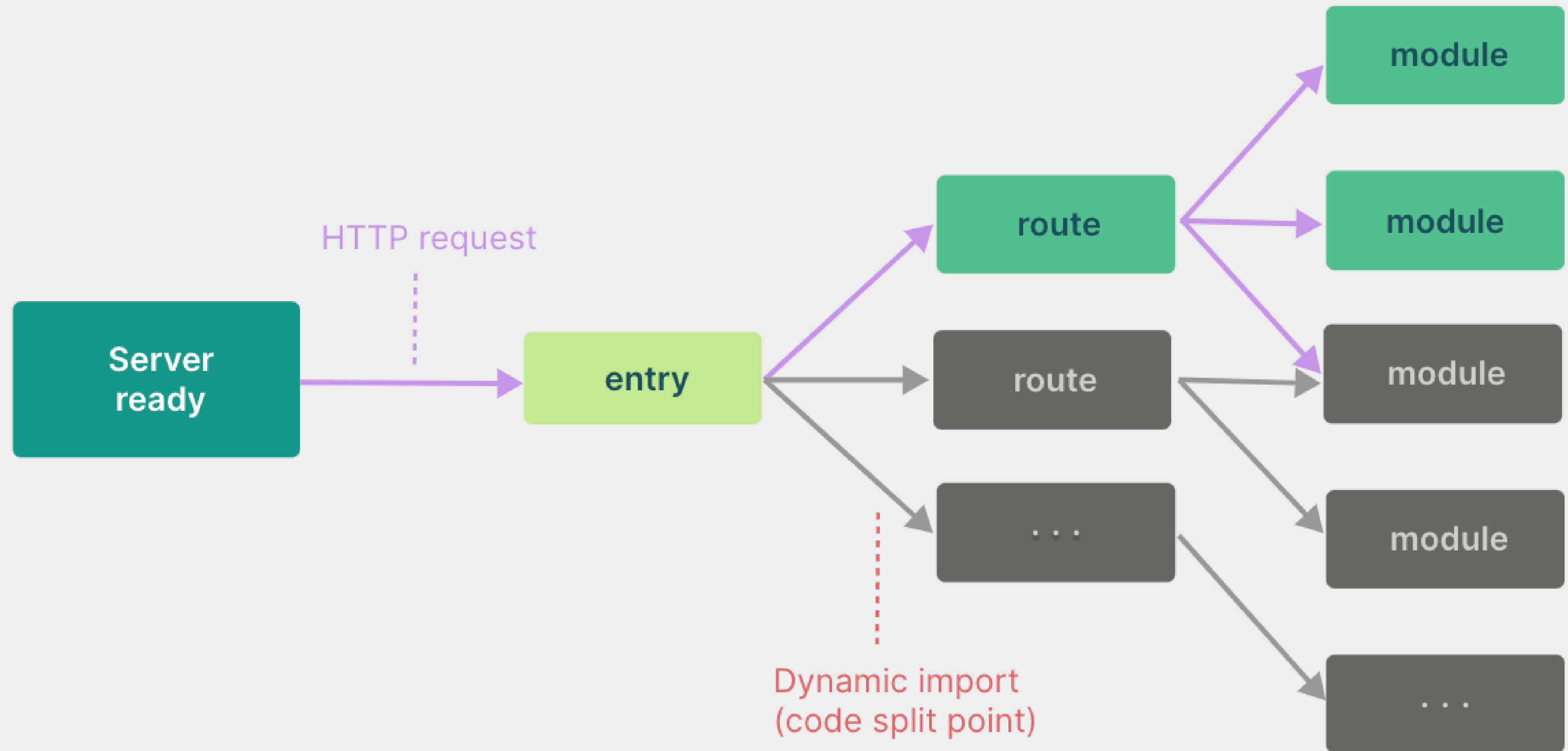
- Vite 以 **原生 ESM** 方式提供源码。

让浏览器接管了打包程序的部分工作：Vite 只需要在浏览器请求源码时进行转换并按需提供源码。根

# Bundle based dev server



# Native ESM based dev server





# 理想和现实

- 尽管原生 ESM 现在得到了广泛支持，但由于嵌套导入会导致额外的网络往返，在生产环境中发布未打包的 ESM 仍然效率低下（即使使用 HTTP/2）。
- 为了在生产环境中获得最佳的加载性能，最好还是将代码进行 tree-shaking、懒加载和 chunk 分割（以获得更好的缓存）

# vite 和 rollup

- Vite 目前的插件 API 与使用 esbuild 作为打包器并不兼容。尽管 esbuild 速度更快,
- 但 Vite 采用了 Rollup 灵活的插件 API 和基础建设, 这对 Vite 在生态中的成功起到了重要作用。

# npx

- an npm package runner
- npm 5.2+ and higher

地址:

- <https://medium.com/@maybekatz/introducing-npx-an-npm-package-runner-55f7d4bd282b>

# create-vite

- <https://github.com/vitejs/vite/blob/main/packages/create-vite/package.json>

# vite 命令

```
"scripts": {  
  "dev": "vite",  
  "build": "vite build",  
  "preview": "vite preview"  
},
```

# vite.config.js

---

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [vue()]
})
```

@vitejs/plugin-vue



# vite-svg-loader


```
import vue from '@vitejs/plugin-vue'
import svgLoader from 'vite-svg-loader'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [
    vue(),
    svgLoader({
      svgo: false,
    }),
  ],
})
```

# 案例实战 - vite 使用 tailwindcss

- 目前最火的原子样式库
- <https://www.tailwindcss.cn/>

# postcss.config.js

```
module.exports = {  
  ..  
   plugins: {  
    tailwindcss: {},  
    autoprefixer: {},  
  },  
}
```

# tailwind.config.js

```
// tailwind.config.js
module.exports = {
  ...
  content: ["../index.htm
  media: false,
  theme: {
    extend: {},
  },
  variants: {
    extend: {},
  },
  plugins: [],
}
```