

Vue2知识点收尾

Vue2的DOM-DIFF原理

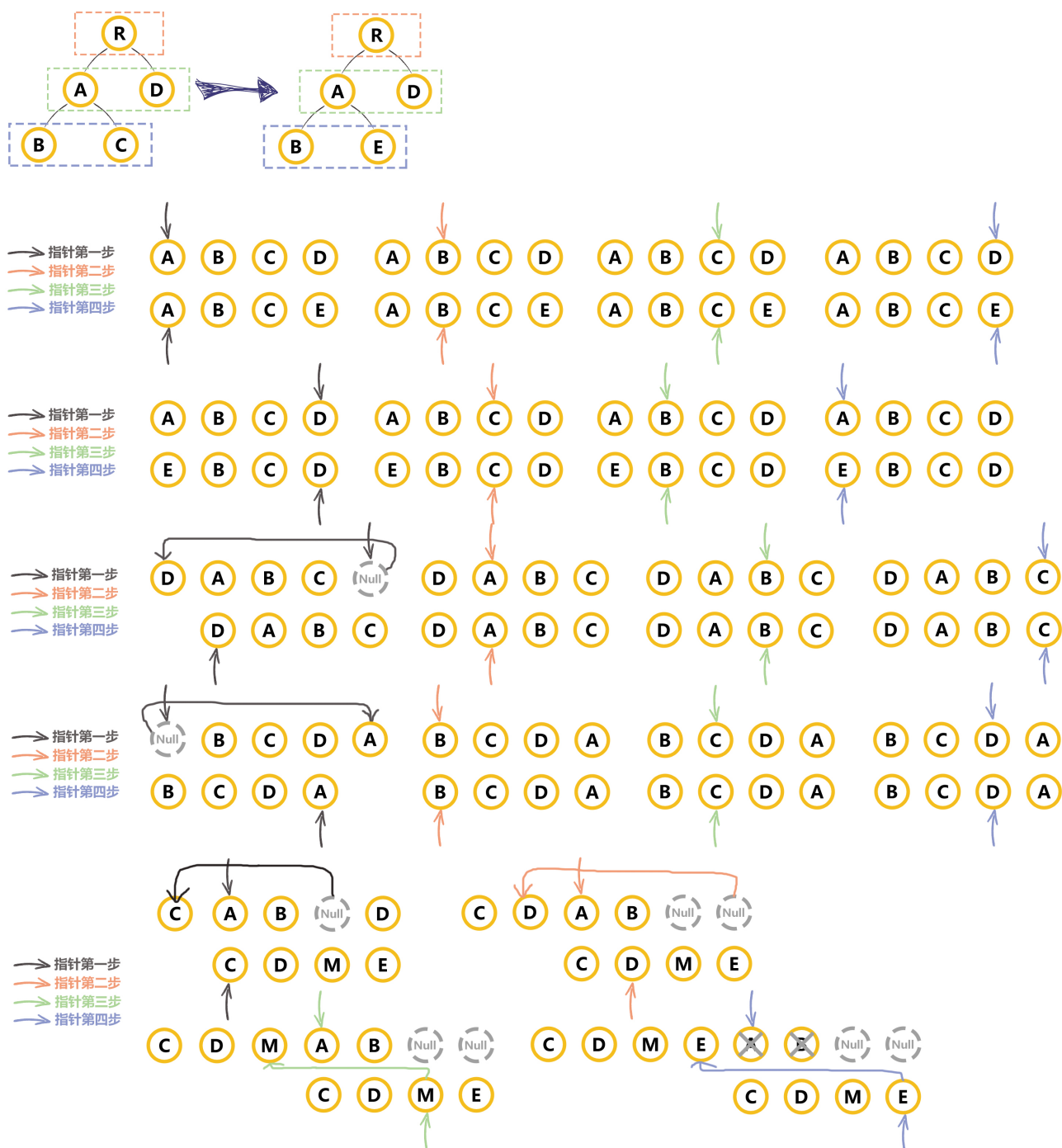
1.1 Diff 概念

Vue 基于虚拟 DOM 做更新。diff 的核心就是比较两个虚拟节点的差异。Vue 的 diff 算法是同级比较，不考虑跨级比较的情况。内部采用深度递归的方式 + 双指针的方式进行比较。

1.2 Diff 比较流程

- 1.先比较是否是相同节点 (isSameVNode)
- 2.相同节点比较属性,并复用老节点（将老的虚拟 dom 复用给新的虚拟节点 DOM）
- 3.比较儿子节点，考虑老节点和新节点儿子的情况
 - 老的没儿子，现在有儿子。直接插入新的儿子
 - 老的有儿子，新的没儿子。直接删除页面节点
 - 老的儿子是文本，新的儿子是文本，直接更新文本节点即可
 - 老的儿子是一个列表，新的儿子也是一个列表
updateChildren
- 4.优化比较：头头、尾尾、头尾、尾头

• 5. 比对查找进行复用



请说明 Vue 中 key 的作用和原理，谈谈你对它的理解？

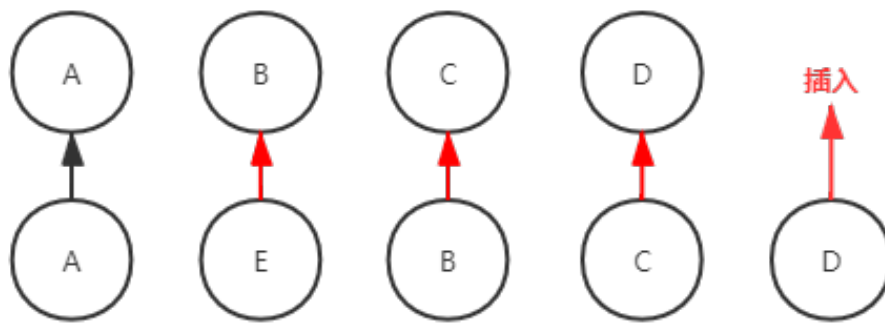
1.1 key 的概念

- `key` 的特殊 attribute 主要用在 Vue 的虚拟 DOM 算法，在新旧 nodes 对比时辨识 VNodes。如果不使用 `key`，Vue 会使用一种最大限度减少动态元素并且尽可能的尝试就地修改/复用相同类型元素的算法。
- 当 Vue 正在更新使用 `v-for` 渲染的元素列表时，它默认使用“就地更新”的策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染

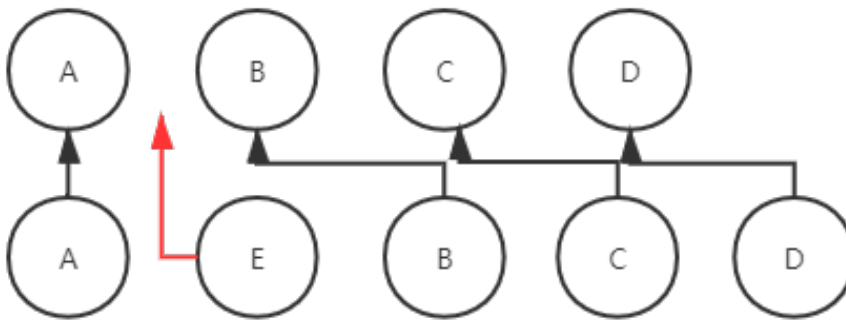
1.2 key 的作用

- Vue 在 patch 过程中通过 `key` 可以判断两个虚拟节点是否是相同节点。（可以复用老节点）
- 无 `key` 会导致更新的时候出问题
- 尽量不要采用索引作为 `key`

无key插入E元素



有key插入E元素



1.3 问题示例

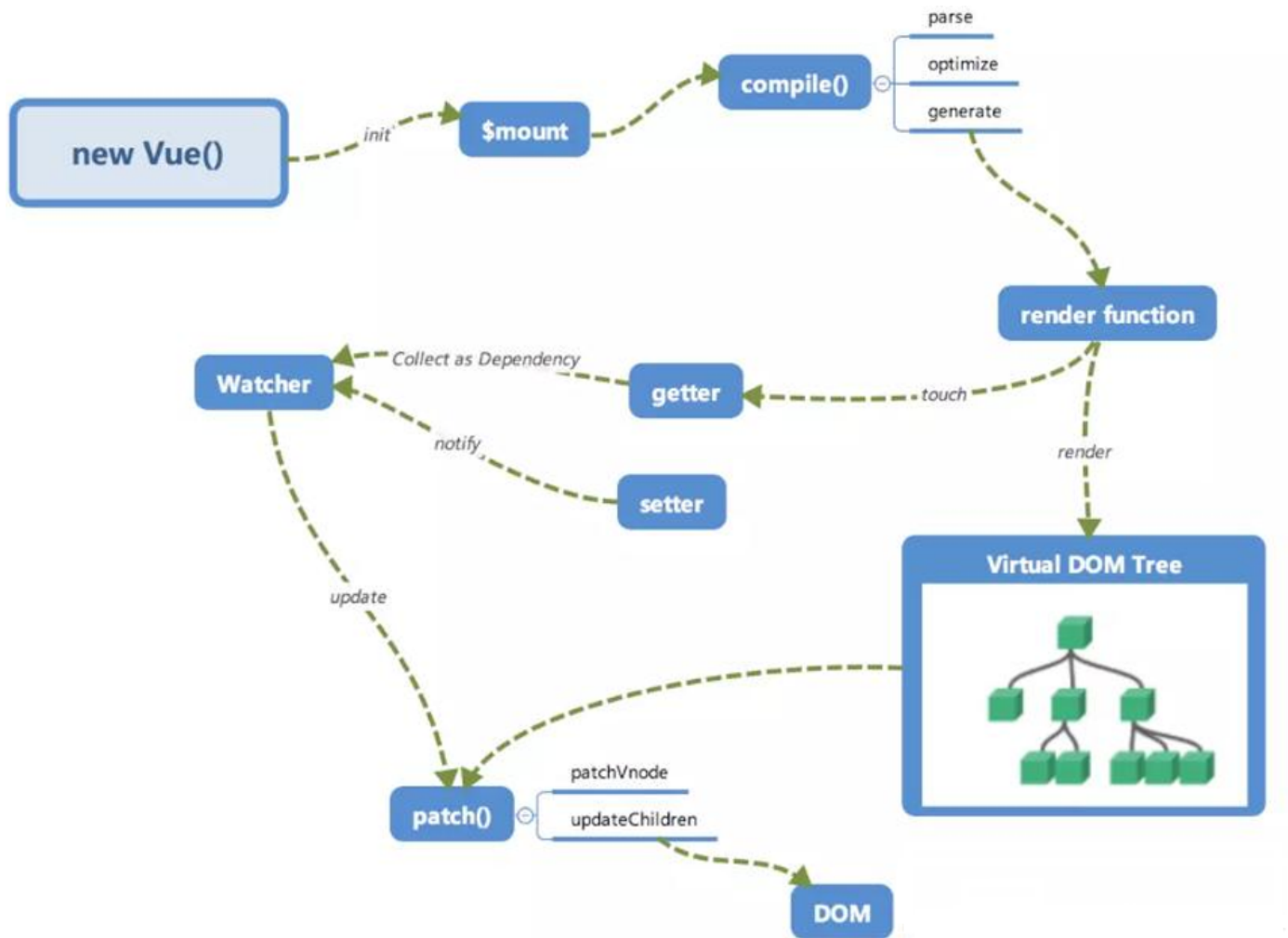
```
<script
src="https://cdn.jsdelivr.net/npm/vue@2">
</script>
<div id="app">
  <li v-for="item in list" :key="item">
    <input type="checkbox"> {{item}}
  </li>
  <button @click="add">增加</button>
</div>
<script>
  const vm = new Vue({
```

```
    el: '#app',
    data: {
      list: [1, 2, 3, 4]
    },
    methods: {
      add() {
        this.list.unshift(Math.random())
      }
    }
  })
</script>
```

Vue 中如何进行依赖收集?

1.1 依赖收集的流程

- 每个属性都拥有自己的 `dep` 属性，存放他所依赖的 `watcher`，当属性变化后会通知自己对应的 `watcher` 去更新
- 默认在初始化时会调用 `render` 函数，此时会触发属性依赖收集 `dep.depend`
- 当属性发生修改时会触发 `watcher` 更新 `dep.notify()`



```

class Dep {
  constructor() {
    this.subs = new Set;
  }
  depend() {
    this.subs.add(Dep.target); // 让属性记住这
    ↑watcher
  }
  notify() {
    this.subs.forEach(watcher =>
watcher.update()); // 通知记住的watcher更新
  }
}
  
```

```

}
class Watcher{
  constructor(fn) {
    this.getter = fn;
    this.get();
  }
  get() { // 第一次渲染
    Dep.target = this;
    this.getter();
    Dep.target = null;
  }
  update() { // 数据变化后更新
    this.get();
  }
}

function defineReactive(obj, key, value) {
  const dep = new Dep();
  Object.defineProperty(obj, key, {
    get() {
      if (Dep.target) { // 说明是在watcher中
        // 访问的属性
        dep.depend()
      }
      return value;
    },
    set(newValue) { // 如果设置的是一个对象那么
      // 会再次进行劫持

```

```

        if (newValue === value) return
        observe(newValue);
        value = newValue
        dep.notify();
    }
})
}
function isObject(value) {
    return typeof value === 'object' && value
    !== null;
}
function observe(value) {
    if(!isObject(value)){
        return;
    }
    Object.keys(value).forEach(key=>{ // 要使用
defineProperty重新定义
        defineReactive(value,key,value[key]);
    });
}
const state = {name:'jw'}
observe(state); // 观测状态，在组件渲染时使用此状态
function render() {
    console.log(state.name)
}
new Watcher(render);

```


Vite

介绍

Vite（法语意为 "快速的"，发音 `/vit/`，发音同 "veet"）是一种新型前端构建工具，能够显著提升前端开发体验。它主要由两部分组成：

- 一个开发服务器，它基于原生 **ES 模块** 提供了丰富的内建功能，如速度快到惊人的**模块热更新（HMR）**。（在开发环境中会基于ES模块启动一个本地服务，优势是可以进行按需加载。并且可以实时预览开发的内容）
- 一套构建指令，它使用 [Rollup](#) 打包你的代码，并且它是预配置的，可输出用于生产环境的高度优化过的静态资源。
（生产环境是基于rollup来进行打包，提供了内置配置，最终打包出来的资源小。可直接部署到服务器上）

Webpack VS Vite

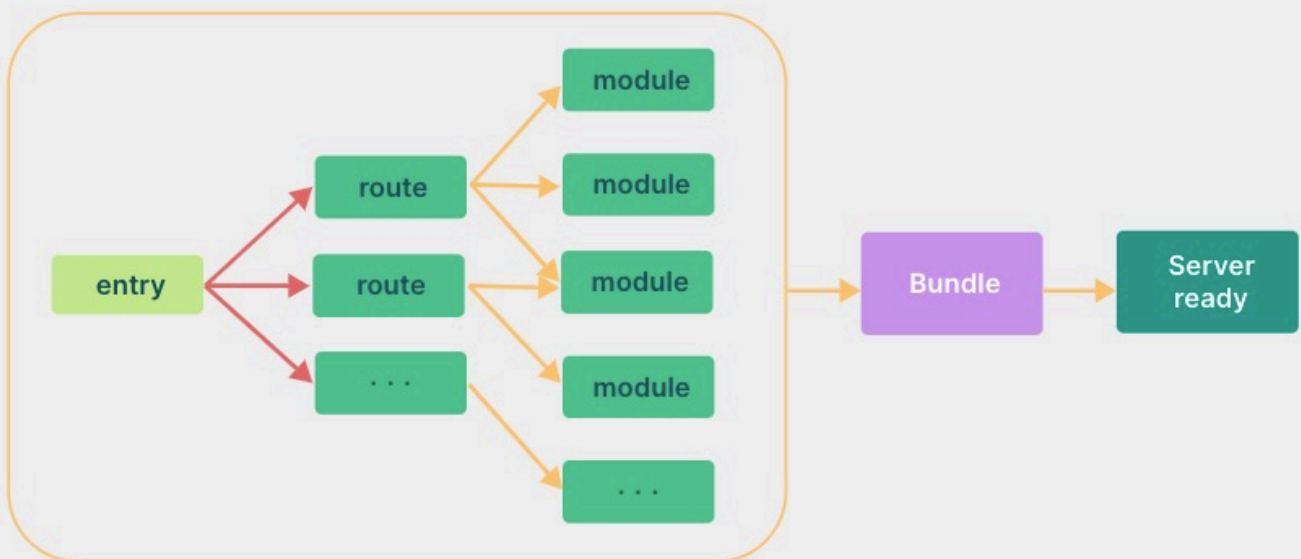
开发环境对比

- Webpack：冷启动（首次启动）开发服务器时，需要构建完整的应用，之后才能提供开发服务。（大型项目启动时非常缓慢，后续更新也非常缓慢） Webpack中大量插件由第三方编写，可能会有性能差等问题。

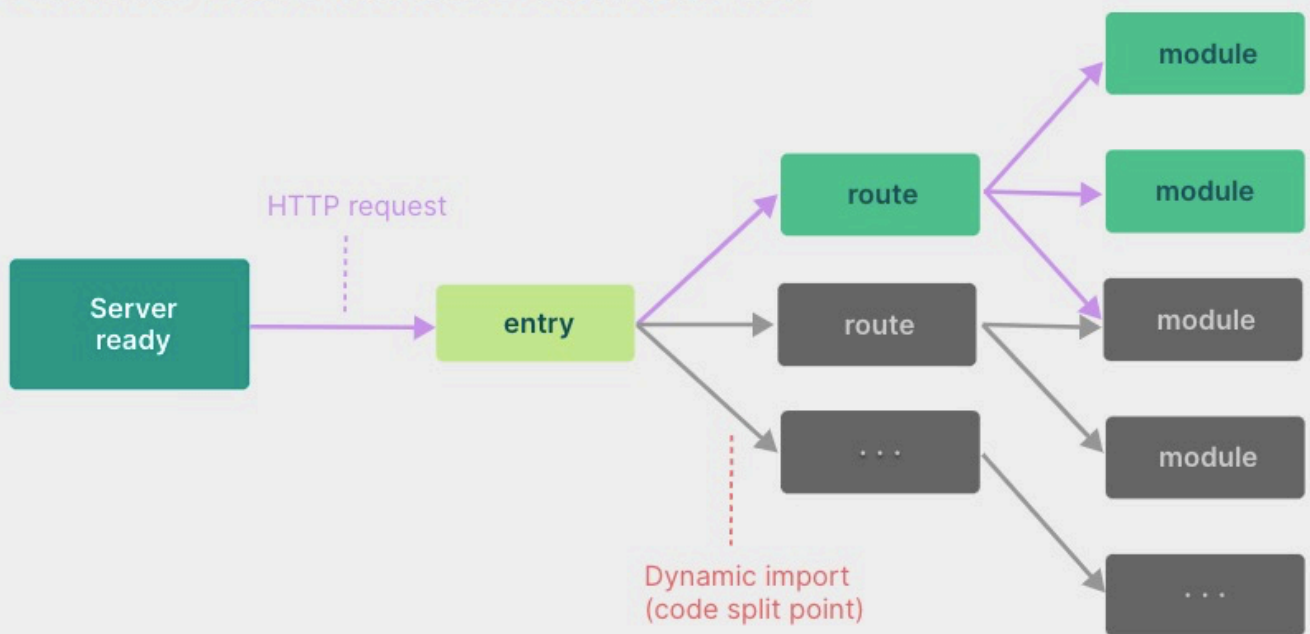
- Vite: 冷启动时, 不需要等待构建完整的应用 (vite支持预构建) vite使用esbuild对第三方模块进行预构建, 将非ESM规范的代码转换为ESM规范的代码, 同时可以将多个文件资源合并成一个, 减少http请求。(Vite是基于ESModule的) 在服务启动后浏览器会根据入口文件需要发送相应请求, 获取资源时对代码进行转化等操作。

快速的冷启动、实时模块转换。

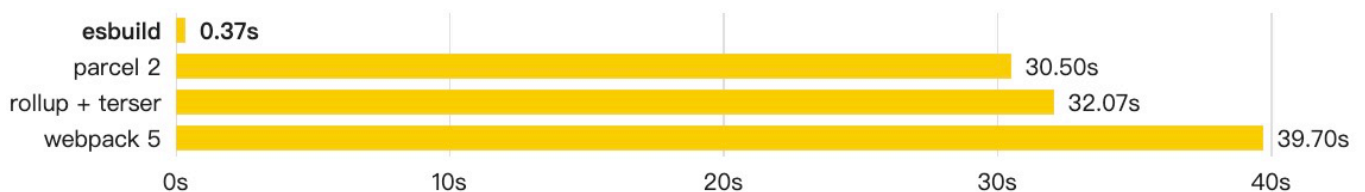
Bundle based dev server



Native ESM based dev server



esbuild



- 基于 Go 语言：ESBuild 是使用 Go 语言编写的构建工具。Go 语言可以充分利用多核CPU 实现并行构建。
- esbuild专注js、ts、jsx的构建和转译，使用自己的算法进行转换和优化，相比其他打包工具速度更快。
- esbuild同样也具有插件系统，可以扩展其自身功能。

但是目前esbuild在构建方面表现出色，但是在生产环境中还不够完善。生产环境我们更关心：代码体积、代码分割、代码缓存等。Vite在生产环境中采用Rollup的目的也是认为Rollup 提供了更好的性能与灵活性方面的权衡。

生产环境对比

- Webpack 是一个非常成熟和广泛采用的打包工具，拥有庞大的生态系统和丰富的插件支持，对于大型复杂的项目，Webpack 的成熟生态系统和丰富的功能一定为首选。
- 对于Vite而言，生产采用的是rollup。
 - **体积更小**：Rollup专注于 JavaScript 模块的打包，可以生成更小、更精简的输出文件，它提供了更好的 tree shaking（摇树优化）功能。
 - **更快的构建速度**：Rollup在打包时的速度通常要比 webpack 更快。
 - **针对库的打包**：Rollup更适合于构建独立的库或组件
 - **ES模块的支持**：Rollup天生支持ES模块的语法。
 - **插件系统**：Rollup具有灵活的插件系统。

Vue2 一般采用webpack来处理，如果使用Vite还需要解决很多问题（老项目更不要在折腾了）。Vue3 项目我们一般采用Vite来进行创建项目，当然也可以使用@vue/cli来进行创建。但要注意的是，ESModule需要浏览器天生支持才可以

Vite项目创建

node版本: v16.20.0

```
pnpm create vite
```

- ✓ Project name: ... vue3-lesson
- ✓ Select a framework: › Vue
- ✓ Select a variant: › Customize with create-vue

Vue.js - The Progressive JavaScript Framework

- ✓ Add TypeScript? ... No / Yes
- ✓ Add JSX Support? ... No / Yes
- ✓ Add Vue Router **for** Single Page Application development? ... No / Yes
- ✓ Add Pinia **for** state management? ... No / Yes
- ✓ Add Vitest **for** Unit Testing? ... No / Yes
- ✓ Add an End-to-End Testing Solution? › No

- ✓ Add ESLint **for** code quality? ... No / Yes
- ✓ Add Prettier **for** code formatting? ... No / Yes

目录结构

- `.vscode`: -> extensions.json 推荐安装插件
- `public`: 该目录通常用于存放不需要经过构建处理的静态资源。这些资源在打包过程中不会被处理或修改，直接复制到输出目录中
- `src`: 源代码目录
 - `assets`: 通常用于存放需要经过构建处理的静态资源，这些资源在打包过程中会被构建工具所处理
 - `components`: 项目中的公共组件
 - `router`: 路由配置
 - `views`: 页面及别组件
 - `App.vue`: 项目根组件
 - `main.js`: 入口文件
- `.eslintrc.cjs`: eslint配置文件，检测代码质量
- `.prettierrc.json`: prettier插件，代码格式化
- `index.html`: 静态文件入口文件
- `vite.config.js`: vite的配置文件

Vite基本使用

1. 图片

服务启动时引入一个静态资源会返回解析后的公共路径

```
<template>
  
</template>
```

2. 基础路径

```
export default defineConfig({
  base: 'http://www.fulljs.cn/' // 基础路径
});
```

3. 路径别名

```

```

```
resolve: {
  alias: {
    '@@': fileURLToPath(new URL('./src/assets',
import.meta.url))
  }
}
```

4.css module

```
<script setup>
import style from './style.module.css'
</script>
<template>
  <h1 :class="$style.red"> 嘿嘿</h1>
  <h1 :class="style.green">哈哈</h1>
</template>
<style module>
.red {color: red;}
</style>
```

5.支持less和sass

```
pnpm install less sass -D
```

```
css: {
  preprocessorOptions: {
    scss: {
      additionalData: '@import
"./assets/scss/var.scss";'
    }
  }
}
```



```
<template>
  <h1 class="less">less</h1>
  <h1 class="scss">scss</h1>
</template>
<style lang="less">
  @less: pink;
  .less {color: @less;}
</style>
<style lang="scss">
  $scss: purple;
  .scss {color: $scss;}
</style>
```

6.postcss配置

postcss.config.js

```
module.exports = {
  plugins: [require('autoprefixer')]
}
```

.browserslistrc

```
> 0.1%
```

7.兼容性处理

```
import legacy from '@vitejs/plugin-legacy'
export default defineConfig({
  plugins: [
    vue(),
    vueJsx(),
    legacy() // 在构建过程中生成传统的 ES5 兼容包，以
    支持旧版本的浏览器
  ],
})
```

8.反向代理

```
server: {
  proxy: {
    '/api': {
      target:
"http://jsonplaceholder.typicode.com",
      changeOrigin: true,
      rewrite: path => path.replace(/^\/api/, '')
    }
  }
}
```

```
fetch('/api/posts/').then(res=>json()).then(data
=> {
  console.log(data)
});
```

9.mock数据

```
pnpm i mockjs vite-plugin-mock -D
```

```
import { viteMockServe } from "vite-plugin-
mock";
export default defineConfig({
  plugins: [
    vue(),
    vueJsx(),
    viteMockServe({})
  ]
});
```

```
export default [  
  {  
    url: '/api/get',  
    method: 'get',  
    response: () => {  
      return {  
        code: 0,  
        data: {name: 'jw'}  
      };  
    },  
  }  
]
```

10.压缩选项

```
build: {  
  minify: 'terser', // 使用terser来压缩  
  assetsInlineLimit: 200 * 1024  
}
```

11.组件自动导入

```
pnpm i unplugin-vue-components -D  
pnpm i vant
```

```
import Components from 'unplugin-vue-components/vite';
import { VantResolver } from 'unplugin-vue-components/resolvers';

export default defineConfig({
  plugins: [
    Components({
      resolvers: [VantResolver()], // vant组件解析
    })
  ],
});
```

你知道Vue3哪些新特性？

[Vue3 迁移指南](#)

1.Composition API

- 使用函数的方式编写vue组件。
- 组合式API (响应式API `ref()`、`reactive()`，生命周期钩子`onMounted()`、`onUnmounted()`，依赖注入`inject()`、`provide()`)

- 组合式API并不是函数式编程。

如何看待Composition API 和 Options API

- 在Vue2中采用的是OptionsAPI, 用户提供的data,props,methods,computed,watch等属性 (用户编写复杂业务逻辑会出现反复横跳问题)
- Vue2中所有的属性都是通过`this`访问, `this`存在指向明确问题
- Vue2中很多未使用方法或属性依旧会被打包, 并且所有全局API都在Vue对象上公开。Composition API对 tree-shaking 更加友好, 代码也更容易压缩。
- 组件逻辑共享问题, Vue2 采用mixins 实现组件之间的逻辑共享; 但是会有数据来源不明确, 命名冲突等问题。Vue3 采用CompositionAPI 提取公共逻辑非常方便



将同一个逻辑的相关代码收集在一起，并且可复用。

2.SFC Composition API Syntax Sugar (<script setup>)

- 单文件组合式API语法糖(setup语法糖)
- 让代码更简洁，性能更好（不需要借助代理对象）。

3.Teleport

- 类似于React中的Portal传送门组件，指定将组件渲染到某个容器中。
- 经常用于处理弹窗组件和模态框组件。

```
<button @click="open = true">打开模态框</button>
<Teleport to="body">
  <div v-if="open" class="modal">
    <button @click="open = false">关闭</button>
  </div>
</Teleport>
```

4.Fragments

- Fragment（片段）Vue3中允许组件中包含多个节点。无需引入额外的DOM元素。

5.Emits Component Option

- Vue3中默认绑定的事件会被绑定到根元素上。通过Emits属性可将事件从 `attrs` 中移除。

6.createRenderer API from @vue/runtime-core to create custom renderers

- 提供自定义渲染器，可以在非DOM环境中使用Vue的运行


```
const {createRenderer,h} = Vue
const renderer = createRenderer({
  createElement(element){
    return document.createElement(element);
  },
  setElementText(el,text){
    el.innerHTML = text
  },
  insert(el,container){
    container.appendChild(el)
  }
});
renderer.render(h('h1','hello
world'),document.getElementById('app'))
```

7.SFC State-driven CSS Variables (v-bind in <style>)

- 在css中使用v-bind绑定样式

```
background: v-bind(color);
```

8.SFC <style scoped> can now include global rules or rules that target only slotted content

- 在作用域样式中可以包含全局规则或只针对插槽内容的规则

```
/* 跨组件修改组件内样式 */  
.parent :deep(h1){color:red}  
/* 控制全局样式 */  
:global(.root){width:  
100px;height:100px;background: yellow;}  
/* 控制插槽传递的内容的样式 */  
:slotted(.child){color:red}
```

9.Suspense experimental

- 主要的作用优雅地处理异步组件的加载状态

```
<Suspense>
  <template #default>
    <!-- 可以配合async setup使用 -->
    <AsyncComponent></AsyncComponent>
  </template>
  <template #fallback>
    正在加载异步组件...
  </template>
</Suspense>
```

Vue3对比Vue2的变化?

1.性能优化（更快）：

- 使用了**Proxy**替代Object.defineProperty实现响应式。（为什么？defineProperty需要对属性进行递归重写添加getter及setter 性能差，同时新增属性和删除属性时无法监控变化，需要\$set、\$delete方法。此方法对数组劫持性能差，同时不支持map和set的数据结构。）
- 模板编译优化。给动态节点增添PatchFlag标记；对静态节点进行静态提升；对事件进行缓存处理等。
- Diff算法优化，全量diff算法中采用最长递增子序列减少节点的移动。在非全量diff算法中只比较动态节点，通过PatchFlag标记更新动态的部分。

2. 体积优化（更小）：

- Vue3移除了不常用的API
 - 移除inline-template (Vue2中就不推荐使用)
 - \$on、\$off、\$once （如果有需要可以采用mitt库来实现）
 - 删除过滤器 （可以通过计算属性或者方法来实现）
 - 移除 `.sync` `.native` 修饰符 (`.sync` 通过 `v-model:xxx` 实现, `.native` 为Vue3中的默认行为) 以及不在支持 `keyCode` 作为 `v-on` 修饰符 (`@keyup.13` 不在支持)
 - 移除全局API。Vue.component、Vue.use、Vue.directive (将这些api挂载到实例上)
- 通过构建工具Tree-shaking机制实现按需引入，减少用户打包后体积。

3. 支持自定义渲染器：

- 用户可以自定义渲染API达到跨平台的目的。扩展能力更强，无需改造Vue源码。

4.TypeScript支持：

- Vue3源码采用Typescript来进行重写，对Ts的支持更加友好。

5.源码结构变化：

- Vue3源码采用 monorepo 方式进行管理，将模块拆分到 package目录中，解耦后可单独使用。

Vue3 响应式数据原理

```
const isObject = (val) => val !== null && typeof
val === 'object';
const proxyMap = new WeakMap();
function createReactiveObject(target) {
  if (!isObject(target)) {
    console.warn(`value cannot be made
reactive: ${String(target)}`);
    return target;
  }
  // 经过劫持处理过的，就不在重复处理了
  const existingProxy = proxyMap.get(target);
  if (existingProxy) return existingProxy;
  // 进行数据劫持
```

```
const proxy = new Proxy(target, {
  get: function get(target, key, receiver)
{
    const res = Reflect.get(target,
key);

    if (isObject(res)) {
        return reactive(res);
    }
    return res;
},
  set: function set(target, key, value,
receiver) {
    let oldValue = target[key];
    if (oldValue === value) return;
    const result = Reflect.set(target,
key, value, receiver);
    console.log('渲染')
    return result;
},
});
proxyMap.set(target, proxy);
return proxy;
}
function reactive(target) {
    return createReactiveObject(target);
}
```

```
const state = reactive({ name: 'jw', arr: [1, 2, 3] })  
state.name = '哈哈'  
state.arr[0] = 100;
```

Vue3语法详解

// vue中组件不在基于类来进行实现，2.x 中函数式组件带来的性能提升在 3.x 中已经可以忽略不计，在vue3中只使用有状态的组件。






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

全局属性定义

```
import 'vant/es/toast/style';
import 'vant/es/dialog/style';
import 'vant/es/notify/style';
import 'vant/es/image-preview/style';
import {
  showToast, showDialog, showNotify, showImagePreview
} from 'vant';

export default function installVant(app) {
  // 以前都是通过vue.prototype来进行扩展
  app.config.globalProperties.$toast =
  showToast;
  app.config.globalProperties.$dialog =
  showDialog;
  app.config.globalProperties.$notify =
  showNotify;
  app.config.globalProperties.$imagePreview =
  showImagePreview
}
```

1. 生命周期

Options API	Composition API
<code>beforeCreate</code>	不需要（直接写到setup中）
<code>created</code>	不需要（直接写到setup中）. 
<code>beforeMount</code>	<code>onBeforeMount</code>
<code>mounted</code>	<code>onMounted</code> 
<code>beforeUpdate</code>	<code>onBeforeUpdate</code>
<code>updated</code>	<code>onUpdated</code> 
<code>beforeUnmount</code>	<code>onBeforeUnmount</code> 
<code>unmounted</code>	<code>onUnmounted</code> 
<code>activated</code>	<code>onActivated</code> （keep-alive中使用）
<code>deactivated</code>	<code>onDeactivated</code> （keep-alive中使用）
<code>errorCaptured</code>	<code>onErrorCaptured</code> （错误捕获）

2.ref和computed使用

```
<template>
  <div class="vote-box">
    <div class="header">
      <h2 class="title">Vue3 很简单</h2>
      <span class="num">总人数7人</span>
    </div>
    <div class="main">
```

```

    <p>支持人数:  3人</p>
    <p>反对人数:  4人</p>
    <p>支持比率:  5%</p>
  </div>
  <div class="footer">
    <van-button type="primary"
@click="change( 'sup' )">支持</van-button>
    <van-button type="warning"
@click="change( 'opp' )">反对</van-button>
  </div>
</div>
</template>
<style scoped>
.vote-box{ width: 300px;margin: 0 auto;padding:
10px;border:1px solid #ccc}
.header {display: flex;justify-content: space-
around;align-items: center;}
.main{margin: 10px 0;}
.footer{justify-content: space-between;}
.van-button{margin-right: 20px;}
</style>

```

```

<script>
import { computed, ref } from 'vue'
export default {
  // 组合式API入口
  setup() {

```

```
// 基于原始数据类型，创建响应式对象
const supNum = ref(0)
const oppNum = ref(0)
// 计算属性
const ratio = computed(() => {
  const total = supNum.value + oppNum.value
  return total === 0 ? '--' : ((supNum.value
/ total) * 100).toFixed(2) + '%'
})
// 组件方法
const change = (type) => {
  type === 'sup' ? supNum.value++ :
oppNum.value++
}
return {
  supNum,
  oppNum,
  ratio,
  change
}
}
}
```

</script>

3.watchEffect和watch

```
<script>
import { computed, ref, watchEffect, reactive,
toRaw, toRefs, watch } from 'vue'
export default {
  // 组合式API入口
  setup() {
    // 基于原始数据类型，创建响应式对象
    // const supNum = ref(0)
    // const oppNum = ref(0)
    const state = reactive({ supNum: 3, oppNum:
0, ratio: '--' }) // 这里不能解构使用

    // 默认执行一次，当依赖的值发生变化会再次执行
    // watchEffect(() => {
    //   let total = state.supNum + state.oppNum
    //   state.ratio = total === 0 ? '--' :
((state.supNum / total) * 100).toFixed(2) + '%'
    // })

    // watch(state) watch(()=>{}) watch([])
    const handle = () => {
      let total = state.supNum + state.oppNum
      state.ratio = total === 0 ? '--' :
((state.supNum / total) * 100).toFixed(2) + '%'
    }
  }
}
```

```

    }
    watch([() => state.supNum, () =>
state.oppNum], handle, { immediate: true })
    // 组件方法
    const change = (type) => {
        type === 'sup' ? state.supNum++ :
state.oppNum++
    }
    return {
        ...toRefs(state),
        change
    }
}
</script>

```

4.如何理解reactive、ref、toRef 和 toRefs?

- **reactive**: 将一个普通对象转换为响应式对象。(采用new Proxy 进行实现) 通过代理对象访问属性时会进行依赖收集, 属性更新时会触发依赖更新。
- **ref**: 创建一个包装对象 (Wrapper Object) 将一个简单的值包装成一个响应式对象, 当访问 `value` 属性时会进行依赖收集, 更新 `value` 属性时会触发依赖更新。(采用类访问器实现) 内部是对象的情况会采用 `reactive` 来进行处理

- **toRef**: 创建 `ref` 对象，引用 `reactive` 中的属性。
- **toRefs**: 批量创建 `ref` 对象，引用 `reactive` 中的属性。

5.watch和watchEffect的区别?

- `watchEffect` 立即运行一个函数，然后被动地追踪它的依赖，当这些依赖改变时重新执行该函数。
- `watch` 侦测一个或多个响应式数据源并在数据源变化时调用一个回调函数。

6.其它响应式API

`shallowReactive`、`shallowRef`、`toRaw`、`markRaw`、`isReadonly`、`isReactive`、`isRef`、`isProxy`等。

Vue3 setup

1.setup参数详解

```
<script setup>
import VoteDemo from './components/VoteDemo.vue'
function oppChange(val) {
  console.log('反对-被触发')
}
function supChange() {
```

```

    console.log('支持-被触发')
  }
  const vote = ref(null);
  onMounted(() => {
    console.log(vote.value)
  })
</script>
<template>
  <VoteDemo v-bind="{a:1,b:2}"
  @oppChange="oppChange" @supChange="supChange"
  title="Vue3其实很简单" ref="vote">
    <template v-slot="{title}">
      <slot>这个模版优先级更高 {{ title }} </slot>
    </template>
  </VoteDemo>
</template>

```

```

<script>
import { ref, computed, watch } from 'vue'
export default {
  props: {
    title: {
      type: String
    }
  },
  emits: ['supChange'],
  setup(props, { attrs, slots, emit, expose }) {

```

```
// setup函数的参数
const supNum = ref(0)
const oppNum = ref(0)
const ratio = computed(() => {
  const total = supNum.value + oppNum.value
  return total === 0 ? '--' : ((supNum.value
/ total) * 100).toFixed(2) + '%'
})
const change = (type) => {
  type === 'sup' ? supNum.value++ :
oppNum.value++
}
watch(
  () => oppNum.value,
  (newVal) => {
    // console.log(attrs) // 包含所有得属性和事件（排除props、以及emits的定义）
    attrs.onOppChange(newVal)
  }
)
watch(
  () => supNum.value,
  (newVal) => {
    emit('supChange', newVal)
  }
)
// 获取插槽信息
```



```
console.log(slots.default({}))

// 暴露实例属性
expose({
  a: 1,
  b: 2
})
return {
  title: props.title,
  supNum,
  oppNum,
  ratio,
  change
}
}
}
</script>
<template>
  <div class="vote-box">
    <div class="header">
      <slot :title="title">
        <h2 class="title">{{ title }}</h2>
      </slot>
      <span class="num">总人数{{ supNum + oppNum
}}人</span>
    </div>
  </div>
</template>
```

```
</template>
```

2.组件name属性

命名合并的方式

```
<script>
export default {name: 'vote'}
</script>
```

```
defineOptions({
  name: 'vote',
  inheritAttrs: false,
}) // 3.3那本以上支持
```

2.defineProps 定义属性

```
<VoteDemo title="Vue3其实很简单"></VoteDemo>
```

```
<script setup>
import { reactive, computed, toRefs } from 'vue'
// 1) 属性
const props = defineProps({
  title: {
    type: String,
    default: 'Vue3其实很简单'
  }
})
```

```
  })  
  // 2) 状态  
  const state = reactive({  
    supNum: 10,  
    oppNum: 5  
  })  
  // 3) 方法  
  const change = (type) => {  
    type === 'sup' ? state.supNum++ :  
state.oppNum++  
  }  
  // 4) 计算属性  
  const ratio = computed(() => {  
    let total = state.supNum + state.oppNum  
    return total === 0 ? '--' : (state.supNum /  
total * 100).toFixed(2) + '%'  
  });  
  const {supNum, oppNum} = toRefs(state)  
</script>
```

3.defineEmits 定义事件

```
const state = reactive({
  supNum: 10,
  oppNum: 5
})

const emit = defineEmits(['supChange']);
watch(() => state.supNum, (newVal) => {
  emit('supChange', newVal)
})
```

4.useAttrs 使用属性

```
const attrs = useAttrs();
watch(() => state.oppNum, (newVal) => {
  attrs.onOppChange(newVal)
})
```

5.useSlots

```
import { useSlots } from 'vue'
const slots = useSlots();
```