Vue对比 Vue 和 JSX/TSX 写法

本文主要针对在 Vue3 中写 JSX 感兴趣的朋友，其实就是想与 jsx 结合，代码跨界更有趣.

为什么能写 jsx/tsx

- 主要原因 vue template 或者 jsx/tsx 都会被编译成 vue 运行时需要的 render 函数或者业界称为 h 函数。所以 vite 项目需要一个 `@vitejs/plugin-vue-jsx` 插件作为支持。

需要注意点

- 使用 vite 初始化项目

```sh
pnpm create vite
```

注意:

- 此时的 vite 的版本是3.x版本（4.x已经发布）

输入项目名，选择 vue 模板， 安装 jsx 支持插件

```css
pnpm i @vitejs/plugin-vue-jsx -D
```

修改 vite 配置文件

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

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

定义模板语法说明

类型	说明
vue 模板	基于 HTML 的模板语法
(j/t)sx	JavaScript 的语法扩展

vue/jsx 注释语法

```vue
<template>
    <!-- you vue comment is html comment syntax -->
</template>
```

```jsx
const Ac = () => {
    return <>
        {/* your component inner comment*/}
    </>
}
```

(j/t)sx className 命名在 Vue 只需要学 class 即可

```tsx
import styles from './index.module.css'
const OptJsxSetupComp = defineComponent({
  setup(a1, b1) {
    const a = ref(0)
    return () => (
      <div class={styles.div}></div>
    );
  }
});
```

htmlFor 不需要 html 前缀, 直接写 for 即可

由于 `for` 在 JavaScript 中是保留字，所以 React 元素中使用了 `htmlFor` 来代替。

```html
<script lang="tsx">
export default {
  name: "LabelForm",
  render() {
    return (
      <form action="/user.html">
        <label for="male">Male</label>
        <input type="radio" name="sex" id="male" value="male" />
        <br />
        <label for="female">Female</label>
        <input type="radio" name="sex" id="female" value="female" />
        <br />
        <input type="submit" value="提交" />
      </form>
    );
  },
};
</script>
```

(j/t)sx style 不使用 : 绑定, 使用 {} 进行绑定、简单运算、函数调用等

```tsx
import styles from './index.module.css'
const OptJsxSetupComp = defineComponent({
  setup(a1, b1) {
    const a = ref(0)
    return () => (
      <div style={{color: 'red'}}></div>
    );
  }
});
```

Vue 定义组件方式

◆ 单文件组件方式:

类型	说明
.vue + 导出对象 + template	vue 单文件组件
.vue + 导出对象 + render-jsx	vue 单文件组件 (lang="jsx or tsx")
.vue + setup + template	vue 单文件组件
.vue + setup + setup-jsx	vue 单文件组件 (lang="jsx or tsx")
.vue + setup + render-jsx	vue 单文件组件 (lang="jsx or tsx")

◆ jsx/tsx 文件方式

类型	说明
.(j/t)sx + defineComponent 函数 + render 函数	(j/t)sx 文件
.(j/t)sx + defineComponent 函数 + setup 函数	(j/t)sx 文件

Options 范式与 Composition 范式组件

Options 范式

```js
export default {
    name: '',
    components: {},
    directives: {},
    props: [],
    data(){return {}},
    compouted:{},
    watch: {},
    emit: {},
    expose: {},
```

```
    template: ``,
    render: () => {}, // 支持 jsx 在此处渲染 jsx 内容
    provide: [],
    inject: [],
    mixins: [],
    extends: [],
    created() {},
    mounted() {},
    //...
}
```

composition

◆ 输出对象形式

```js
export default {
    setup(props, context) { // context: attrs, slots, emit, expose
        // your code
        return {},  // 支持 jsx 此处渲染一个函数，此函数返回一个 jsx
    },
}
```

◆ 直接定义脚本形式

```html
<script setup>
    // 变量
    const msg = 'Hello!'
    // 函数
    function log() { console.log(msg) }
</script>
```

(j/t)sx 文件中使用 defineComponent 定义个 setup 组件

◆ (j/t)sx 文件中 setup 返回一个函数，此函数返回一个 (j/t)sx

```tsx
const OptJsxSetupComp = defineComponent({
  setup(props) {
    return () => (
      <div>x</div>
    );
  }
});
```

◆ (j/t)sx 文件中 render 返回一个 (j/t)sx

```tsx
const OptJsxSetupComp = defineComponent({
  render(this) { // this 组件实例， data 可以此处拿到
    return (
      <div>x</div>
    );
```

```
  }
});
```

绑定数据

- vue 模板使用 `{{ data }}`

```vue
<template>
  <div>{{a}}</div>
</template>

<script lang="ts">
export default {
  data() {
    return {a: 1}
  }
}
</script>
```

- vue setup + template + 选项

```vue
<template>
  <div>{{a}}</div>
</template>

<script lang="ts">
import { ref } from 'vue'
export default {
  setup() {
    let a = ref(0)
    return {
      a
    }
  }
}
</script>
```

- vue + script + template + setup 属性

```vue
<template>
  <div>{{a}}</div>
</template>

<script lang="ts" setup>
import { ref } from 'vue'

let a = ref(0)
</script>
```

- (j/t)sx + defineComponent + render 绑定 options data

```tsx
const OptJsxComp = defineComponent({
  data() {
    return {
      a: 23,
    };
  },
  render(this: any) {
    let { a } = this;
    return (
      <div>{a}</div>
    );
  },
});
```

◆ (j/t)sx + defineComponent + setup 绑定 ref data

```tsx
const OptJsxSetupComp = defineComponent({
  setup() {
    const a = ref(0)
    return () => (
      <div> {a.value}</div>
    );
  }
});
```

绑定事件

类型	说明	示例
vue	使用指令 v-on 或者 指令缩写 @ 绑定事件	`@click='handleClick'`
jsx	使用 on +事件名（大写开头）	onClick={handleClick}

注意：vue 听了修饰器语法糖， jsx 中没有需要自己实现

修饰符	处理方式
prevent	event.preventDefault()
stop	event.stopPropagation()

其他事件修饰符自行探索

绑定事件修改响应式数据

vue 基于可变数据进行设计（与 React 不同），可变性标志着，可以直接用 js 原生提供了操作符和方法进行访问和修改，但也不是全部。数组方式仅仅支持了一部分：

- js 原始类型数据直接修改
- js 非原始类型数据，在vue要分为可变和不变两种类型：下面是针对数组的分类：
- 可变

可变数组方式	说明
`push()`	尾部添加
`pop()`	去除尾部
`shift()`	首部去除
`unshift()`	首部添加
`splice()`	裁剪
`sort()`	排序
`reverse()`	翻转

- 不可变

可变数组方式	说明
`filter()`	过滤
`concat()`	连接
`slice()`	切片

获取 props 与 props 约束

- props 的定义约束

```html
<template>
 <div>{{poo}}</div>
</template>

<script lang="ts">
export default {
 name: 'IPropsOpt',
 props: ['poo']
}
</script>
```

- script + setup attr 中定义约束

```html
<script lang="ts" setup>
import { defineProps } from 'vue'
const props = defineProps(['foo'])

</script>

<template>
  <div>{{props.foo}}</div>
</template>
```

◆ tsx + setup

```tsx
import { defineComponent } from 'vue'

type IProps = {
  name: string
}

const II1 = defineComponent({
  setup(props: IProps) {
    const { name } = props;
    return () => {
      return <div>{name}</div>
    }
  }
})

export default II1;
```

◆ tsx + render + options

```tsx
import { defineComponent } from 'vue'

const IC = defineComponent({
  props: ['title'],
  render() {
    return <div>{this.title}</div>
  }
})

export { IC }
```

◆ props 约束

```ts
defineProps({ name: String, agee: Number }) // composition setup
export default { props: { name: String, age: Number } } // options
```

vue 中指令

◆ v-if 使用在 jsx 中使用三元表示

```html 复制代码
<script lang="tsx">
export default {
  name: 'IfVueD',
  data() {
    return {
      isShow: false
    }
  },
  render() {
    return <div>{this.isShow ? 'show' : 'hidden'}</div>
  }
}
</script>
```

- v-show 使用控制 style 的方式表示

```tsx 复制代码
<script lang="tsx">
export default {
 name: 'IShow',
 data() {
   return {
     isShow: false
   }
 },
 render() {
   return <div style={{ display: this.isShow ? 'show' : 'hidden'}}>show</div>
 }
}
</script>
```

- v-html/v-text 与 jsx 的 ```dangerouslySetInnerHTML={{__html: html}} 类似` 注意(这里是危险的)

```html 复制代码
<script lang="tsx">
export default {
  name: 'render html',
  data() {
    return {
      isShow: false,
      html: '<div style="color: red">this is false</div>'
    }
  },
  render() {
    return <div innerHTML={this.html}></div>
  }
}
</script>
```

注意这里是：`innerHTML` 与 React 不一样 dangerouslySetInnerHTML。

- v-for 使用 map

```html 复制代码
<script lang="tsx">
export default {
  name: 'IfVueD',
  data() {
    return {
      list: [10,20,30,40]
    }
  },
  render() {
    return <div>
      {this.list.map((l, i) => {
        return <div key={i}>{l}-{i}</div>
      })}
    </div>
  }
}
</script>
```

◆ v-on 事件监听 使用 `onClick`，修饰符没有语法糖需要单独自己处理

```html 复制代码
<script lang="tsx">
export default {
  name: "IfVueD",
  data() {
    return {
      list: [10, 20, 30, 40],
    };
  },
  render() {
    return (
      <div>
        {this.list.map((l, i) => {
          return (
            <div
              onClick={() => { // onXxxx 事件绑定
                alert(i);
              }}
              key={i}
            >
              {l}-{i}
            </div>
          );
        })}
      </div>
    );
  },
};
</script>
```

◆ v-model 有语法糖实现。修饰符需要自己处理

```html
<script lang="tsx">
export default {
  name: 'Model',
  data() {
    return {
      content: ''
    }
  },
  render() {
    return <div>
      <div>{this.content}</div>
      <input v-model={this.content} />
    </div>
  }
}
</script>
```

◆ v-slot 与 React 的 children 要区分开发(以下是默认插槽，具名插槽)

render 函数中使用 this 实例获取 ￥slots 对象

```html
<script lang="tsx">
export default {
  name: 'SlotDeafult',
  render() {
    return <div>
      {this.$slots.header!()}
      {this.$slots.default!()}
       {this.$slots.footer!()}
    </div>
  }
}
</script>

// 父组件使用
<SlotDefaultVue>
  <template #header>
    <div>header</div>
  </template>
  <div>sdf</div>
  <template #footer>
    <div>footer</div>
  </template>
</SlotDefaultVue>
```

以下是 setup 的写法

```html
<script lang="tsx">
export default {
  name: 'SlotDeafult',
  setup(props, { slots }) {
```

```
    return () => <div>
      {slots.header!()}
      {slots.default!()}
       {slots.footer!()}
    </div>
  }
}
</script>
```

组件中生命周期

组件中生命周期定义方法

```html
<script lang="tsx">
export default {
  created() {
    console.log('created')
  },
  mounted() {
    console.log('mounted')
  },
  beforeUnmount() {
    console.log('beforeUnmount')
  }
}
</script>
```

composition setup 范式中

```html
<script lang="tsx" setup>
import { onBeforeMount, onMounted, onBeforeUpdate } from 'vue';
onMounted(() => {
  console.log("onMounted")
})
onBeforeMount(() => {
  console.log("onBeforeMount")
})

onBeforeUpdate(() => {
  console.log("onBeforeUpdate")
})
</script>
```

计算属性

computed options 变化 => composition computed()

```html
<script lang="tsx" setup>
import { computed, ref } from 'vue'
```

```
const count = ref(0)

const myCount = computed(() => count.value + 10)
</script>

<template>
  <div>
    {{count}}
    {{myCount}}
  </div>
</template>
```

侦听器

watch => watch

```html
<script lang="tsx" setup>
import { watch, ref } from 'vue';

let count = ref(0);

watch(count, (nc, oc) => {
  if(nc < 3) {
    count.value = 5
  } else {
    count.value += 1
  }
})
</script>
```

小结

- vue 定义组件的方式多种多样，本文主要以文件形式讲解单文件组件和tsx、jsx 文件如何使用 vue 组件，当然文章是抛砖引玉。。
- render 方式渲染 jsx 使用 this 对象来获取 vue 中的示例属性
- setup 渲染 jsx 使用 setup 的第一个参数是 props, 第二个参数 ctx 来获取对应参数

用 vue 写一个 todos 应用

- todo-vue3-tsx 以下是实现的一个 vue + tsx 版本的基本功能，代码也在优化当中。

参考

- Vue 官方文档 jsx-tsx
- Vite Vue JSX 插件(使用 babel)
- JSX & TSX
- FaceBook JSX
- JSX 简介