

# 主要是记录一次解决交互卡顿的方法

- 实际业务场景：
  - 点击左侧标签，例如标题0、标题1、，中间的对应标题的checkboxgroup会滚动到最顶，而中间checkbox的选择会和最右侧的tags选择联动，两者始终保持一致
- 伪代码
  - 伪代码模拟了关键的功能，点击左侧标签改为高亮对应的分组，原理是一样的



# 优化效果

- 优化前

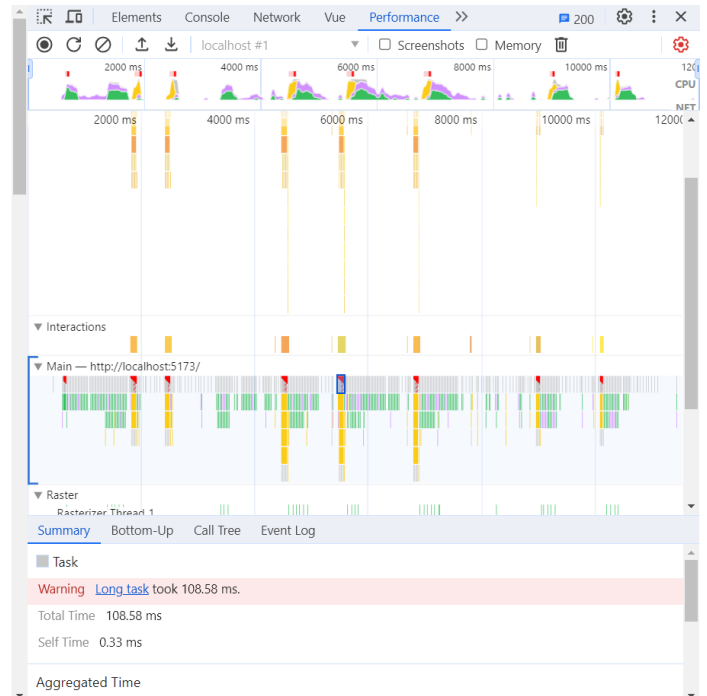
- 业务场景

- 代码还有其他功能，例如动态的tooltip（涉及js计算），以及更加多的嵌套关系，真实的性能肯定比伪代

码要更差一些，每次点击左边标签或选择checkbox，大概会产生200ms左右长任务

- 伪代码

- 100ms+



- 优化后

- 业务场景

- 每次交互，产生的任务是50ms+

- 伪代码

- 10ms+

优化方法一：拆分组件，修改入参

ListComponentAfterOptimize.vue

分类标题 2

标题0

option0-0 option0-1 option0-2 option0-3 option0-4 option0-5 option0-6 option0-7

option0-8 option0-9 option0-10 option0-11 option0-12 option0-13 option0-14

option0-15 option0-16 option0-17 option0-18 option0-19 option0-20 option0-21

option0-22 option0-23 option0-24 option0-25 option0-26 option0-27 option0-28

option0-29 option0-30 option0-31 option0-32 option0-33 option0-34 option0-35

option0-36 option0-37 option0-38 option0-39 option0-40 option0-41 option0-42

option0-43 option0-44 option0-45 option0-46 option0-47 option0-48 option0-49

option0-50 option0-51 option0-52 option0-53 option0-54 option0-55 option0-56

option0-57 option0-58 option0-59 option0-60 option0-61 option0-62 option0-63

option0-64 option0-65 option0-66 option0-67 option0-68 option0-69 option0-70

option0-71 option0-72 option0-73 option0-74 option0-75 option0-76 option0-77

option0-78 option0-79 option0-80 option0-81 option0-82 option0-83 option0-84

option0-85 option0-86 option0-87 option0-88 option0-89 option0-90 option0-91

option0-92 option0-93 option0-94 option0-95 option0-96 option0-97 option0-98

option0-99

标题1

option1-0 option1-1 option1-2 option1-3 option1-4 option1-5 option1-6 option1-7

option1-8 option1-9 option1-10 option1-11 option1-12 option1-13 option1-14

option1-15 option1-16 option1-17 option1-18 option1-19 option1-20 option1-21

option1-22 option1-23 option1-24 option1-25 option1-26 option1-27 option1-28

option1-29 option1-30 option1-31 option1-32 option1-33 option1-34 option1-35

option1-36 option1-37 option1-38 option1-39 option1-40 option1-41 option1-42

option1-43 option1-44 option1-45 option1-46 option1-47 option1-48 option1-49

option1-50 option1-51 option1-52 option1-53 option1-54 option1-55 option1-56

option1-57 option1-58 option1-59 option1-60 option1-61 option1-62 option1-63

option1-64 option1-65 option1-66 option1-67 option1-68 option1-69 option1-70

option1-71 option1-72 option1-73 option1-74 option1-75 option1-76 option1-77

option1-78 option1-79 option1-80 option1-81 option1-82 option1-83 option1-84

option1-85 option1-86 option1-87 option1-88 option1-89 option1-90 option1-91

option1-92 option1-93 option1-94 option1-95 option1-96 option1-97 option1-98

option1-99

标题2

option2-0 option2-1 option2-2 option2-3 option2-4 option2-5 option2-6 option2-7

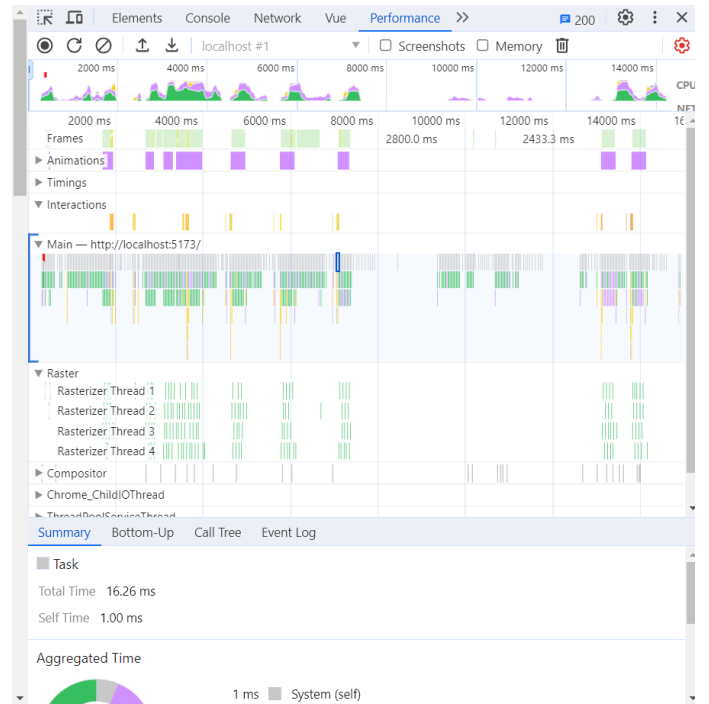
option2-8 option2-9 option2-10 option2-11 option2-12 option2-13 option2-14

option2-15 option2-16 option2-17 option2-18 option2-19 option2-20 option2-21

option2-22 option2-23 option2-24 option2-25 option2-26 option2-27 option2-28

option2-29 option2-30 option2-31 option2-32 option2-33 option2-34 option2-35

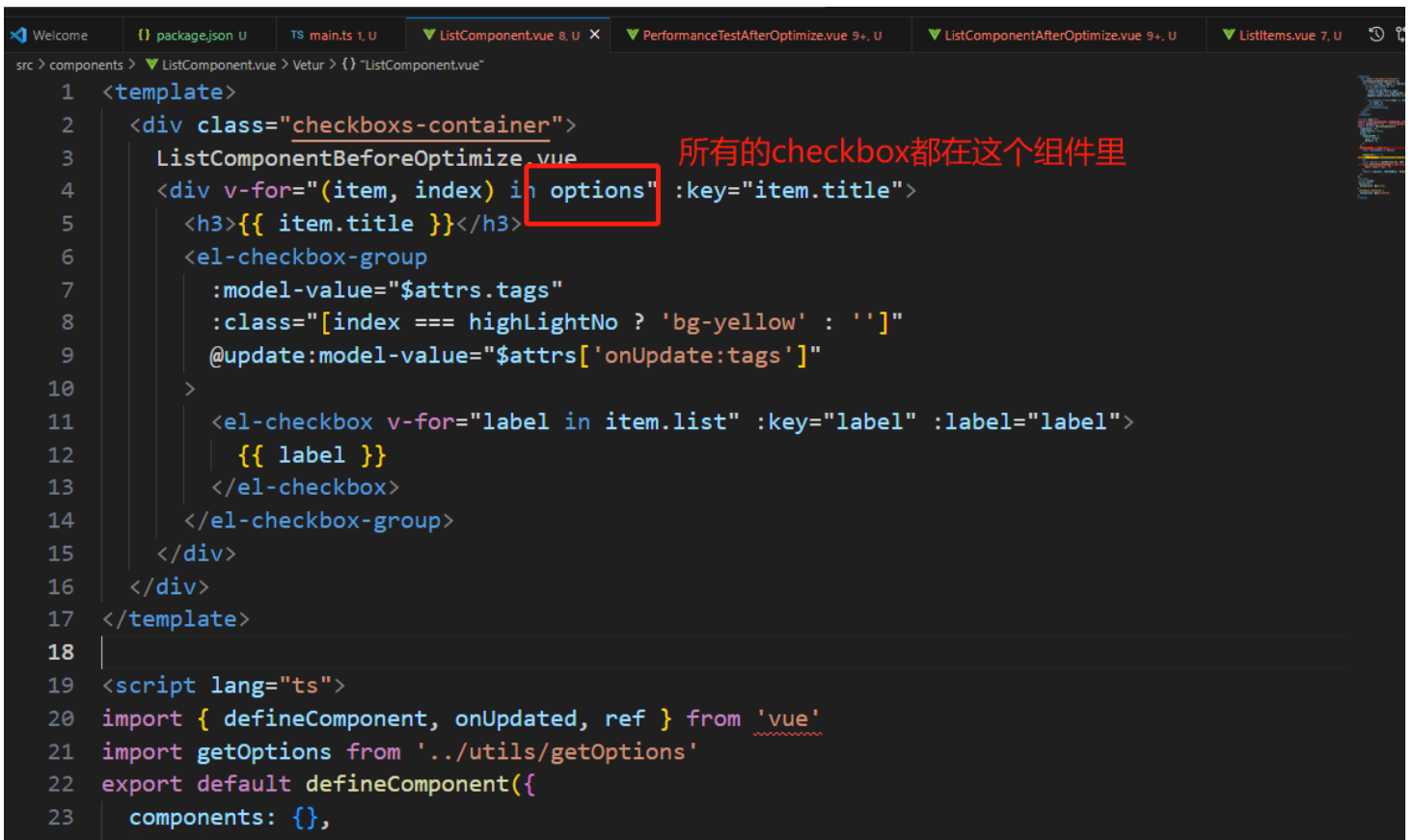
option2-36 option2-37 option2-38 option2-39 option2-40 option2-41 option2-42



# 优化过程

## 未优化时组件的结构

```
<template>
  <h3>未优化的时候</h3>
  <div class="flex-container">
    <div class="highlight-index">
      高亮标题
      <el-input-number v-model="highLightNo" />
    </div>
    <ListComponent v-model:tags="tags" :high-light-no="highLightNo" />
  </div>
  <div class="tags-display">
    <el-tag
      v-for="tag in tags"
      :key="tag"
      class="mx-1"
      closable
      @close="handleClose(tag)"
    >
      {{ tag }}
    </el-tag>
  </div>
</div>
</template>
```

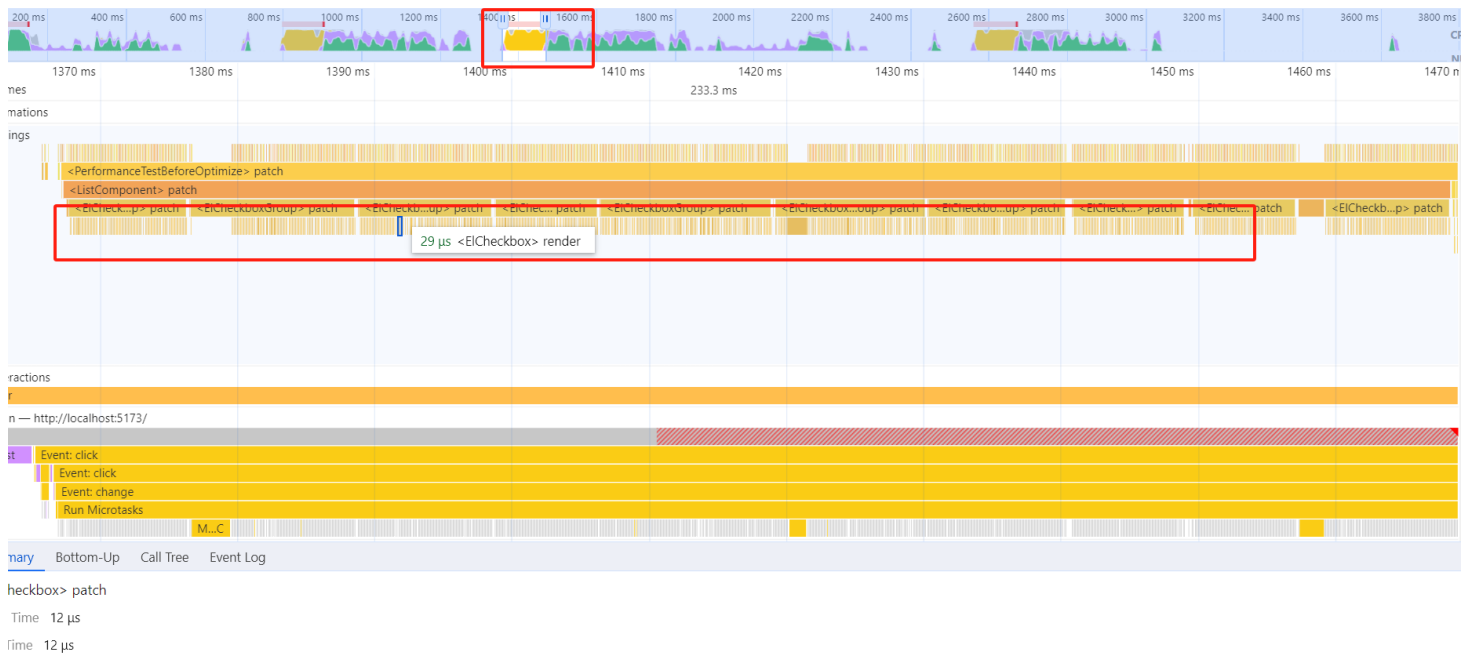


```
1 <template>
2   <div class="checkboxs-container">
3     ListComponentBeforeOptimize.vue
4     <div v-for="(item, index) in options" :key="item.title">
5       <h3>{{ item.title }}</h3>
6       <el-checkbox-group
7         :model-value="$attrs.tags"
8         :class="[index === highLightNo ? 'bg-yellow' : '']"
9         @update:model-value="$attrs['onUpdate:tags']"
10      >
11       <el-checkbox v-for="label in item.list" :key="label" :label="label">
12         {{ label }}
13       </el-checkbox>
14     </el-checkbox-group>
15   </div>
16 </div>
17 </template>
18
19 <script lang="ts">
20 import { defineComponent, onUpdated, ref } from 'vue'
21 import getOptions from '../utils/getOptions'
22 export default defineComponent({
23   components: {},
24   ...
25 })
```

## • 传参方式

- 最左边的选择高亮的标题，通过highLightNo这个prop传给ListComponent组件
  - 最右边的tags通过tags这个prop传给ListComponent，也就是说，这两个prop都会导致ListComponent这个组件的更新
- ListComponent这个组件里面存在大量的el-checkbox，每次任何prop的改变，ListComponent组件更新，更新的时候会把子组件全部都render->patch一次，因为checkbox的数量很多，每一个checkbox的render->patch时间很短。就造成了长任务

如下图



## 优化方式1

如果我们不讨论真实业务场景，仅讨论伪代码里面label的显示样式，也就是不需要**自定义checkbox里面label的场景**，其实是不需要用到slot的方式的，

只需要像黄色圈起的地方这样使用即可

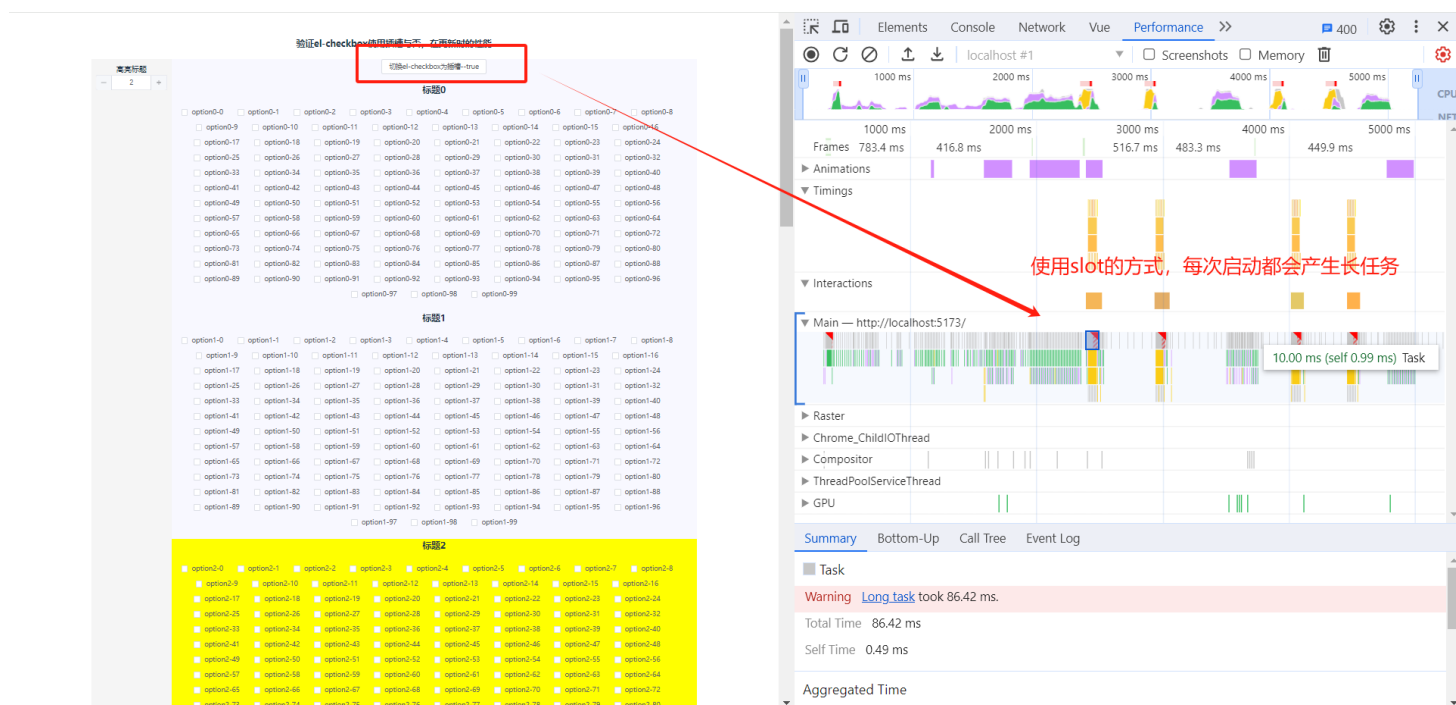
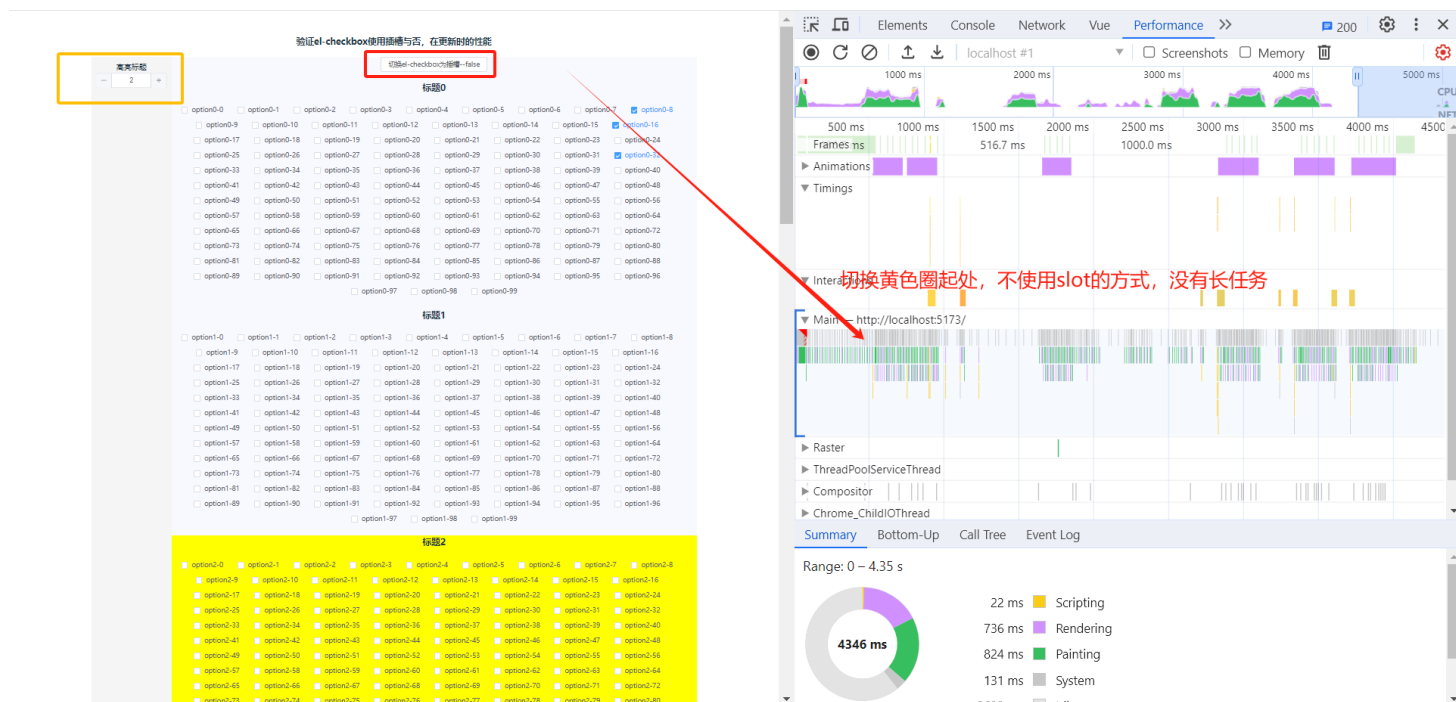
```
8      :key="item.title"
9      :class="[index === highLightNo ? 'bg-yellow' : '']"
10    >
11      <h3>{{ item.title }}</h3>
12      <template v-if="useSlotOrNot">
13        <el-checkbox v-for="label in item.list" :key="label" :label="label">
14          {{ label }}
15        </el-checkbox>
16      </template>
17      <template v-if="!useSlotOrNot">
18        <el-checkbox v-for="label in item.list" :key="label" :label="label" />
19      </template>
20    </div>
```

平时很多使用element的组件的时候也没有注意问题，但其实这样会造成性能的巨大差异

请看main.ts这个引用的文件，这个文件省略了右边tags，因为这个文件仅用来说明slot的影响，不需要右边tags的场景

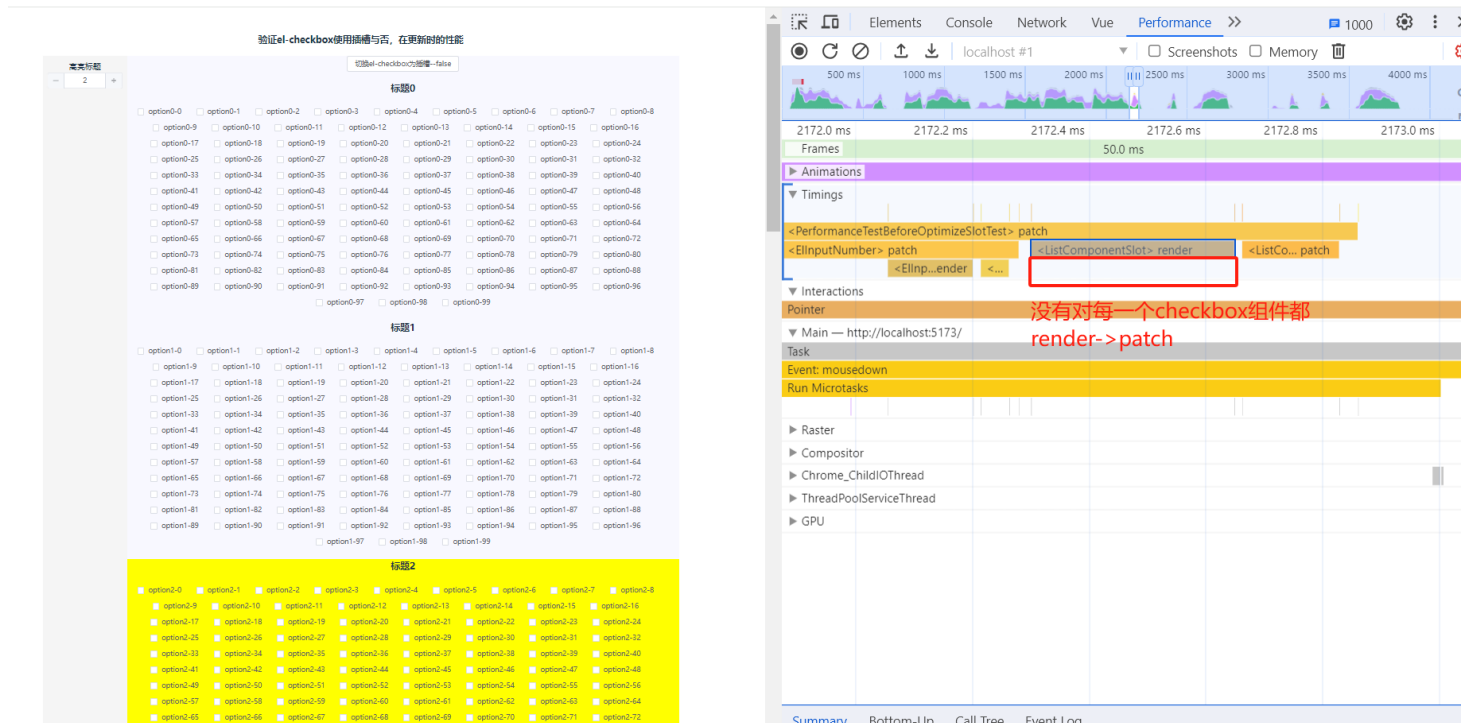
```
10 // 优化方式2
11 // import App from './PerformanceTestAfterOptimize2.vue'
12 // el-checkbox不需要使用slot的情况下，使用slot和非slot的性能差异
13 import App from './PerformanceTestBeforeOptimizeSlotTest.vue'
14
15 const app = createApp(App)
```

我们实验一下看看差异





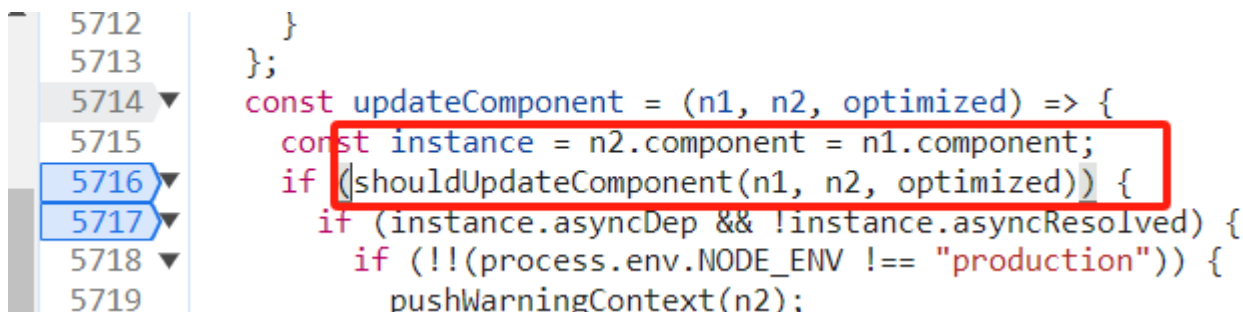
主要的区别是 不使用插槽的时候没有对每一个checkbox的render->patch



## 什么使用插槽的方式会导致elcheckbox全更新的

- 使用插槽

因为使用插槽，特别是`{{xxx}}`这种动态插槽，在编译模版时，会把这个组件的`patchFlag`设为1204，即使在输入高亮标题的时候，对于`el-checkbox`这个组件来说，`props`完全没有变化，但是`patchFlag`在`shouldUpdateComponent`里面还是任务需要更新，如下图，最后每一个`el-checkbox`都需要更新了



```

return vnode.shapeFlag & (6 | 1) || vnode.type === Comment;
};
function shouldUpdateComponent(prevVNode, nextVNode, optimized) {
  const { props: prevProps, children: prevChildren, component } =
  const { props: nextProps, children: nextChildren, patchFlag } =
  const emits = component.emitsOptions;
  if (!(process.env.NODE_ENV !== "production") && (prevChildren
    return true;
  }
  if (nextVNode.dirs || nextVNode.transition) {
    return true;
  }
  if (optimized && patchFlag >= 0) {
    if (patchFlag & 1024) {
      return true;
    }
    if (patchFlag & 16) {
      if (!prevProps) {
        return !!nextProps;
      }
      return hasPropsChanged(prevProps, nextProps, emits);
    } else if (patchFlag & 8) {
      const dynamicProps = nextVNode.dynamicProps;
      for (let i = 0; i < dynamicProps.length; i++) {
        const key = dynamicProps[i];
        if (nextProps[key] !== prevProps[key] && !isEmitListener(
          return true;
        }
      }
    }
  }
  return false;
}

```

shouldUpdateComponent

2 matches



Cancel

- 不使用插槽

patchFlag是8，并进行前后prop的对比，最后发现prop没变化，于是返回false，所以el-checkbox这个组件并没有再次更新

```

1003 function shouldUpdateComponent(prevVNode, nextVNode, optimized) {
1004   const { props: prevProps, children: prevChildren, component } =
1005   const { props: nextProps, children: nextChildren, patchFlag } =
1006   const emits = component.emitsOptions;
1007   if (!(process.env.NODE_ENV !== "production") && (prevChildren
1010   if (nextVNode.dirs || nextVNode.transition) {...}
1013   if (optimized && patchFlag >= 0) {
1014     if (patchFlag & 1024) {
1015       return true;
1016     }
1017     if (patchFlag & 16) {
1018       if (!prevProps) {
1019         return !!nextProps;
1020       }
1021       return hasPropsChanged(prevProps, nextProps, emits);
1022     } else if (patchFlag & 8) {
1023       const dynamicProps = nextVNode.dynamicProps;
1024       for (let i = 0; i < dynamicProps.length; i++) {
1025         const key = dynamicProps[i];
1026         if (nextProps[key] !== prevProps[key] && !isEmitListener(
1027           return true;
1028         }
1029       }
1030     }
1031   } else {...}
1048   return false;
1049 }

```

仅仅prop真的变化了才返回  
 true, 代表需要更新, 否则返回  
 false, 代表不需要更新

这就是为什么使用slot与否性能差距比较大的原因了

顺带说个有趣的点, 就是我们对组件绑定方法的方式不同, 也会导致组件的更新不同

下面 @update:tags 的绑定, 如果其他prop均没有变化, 上面的绑定方式不会导致组件的更新, 下面画红框的就会 (这个我也没研究为什么, 记入to-do-list吧吧吧吧吧吧吧)

```

ListComponentAfterOptimize.vue
<div v-for="(item, index) in options" :key="item.title">
  <ListItems
    :item="item"
    :index="index"
    :should-high-light="highLightNo === index"
    :tags="$attrs.tags[item.title]"
    @update:tags="handleCheckChanges"
    @update:tags="(index) => handleCheckChanges(index, item.title)"
  />
</div>

```

在真实的业务场景下，需要自定义插槽的使用场景颇多，所以还需要其他方式解决问题

**这次的优化核心就是重新安排每个组件的颗粒度，使tags和highLightNo这两个props对组件的影响减少**

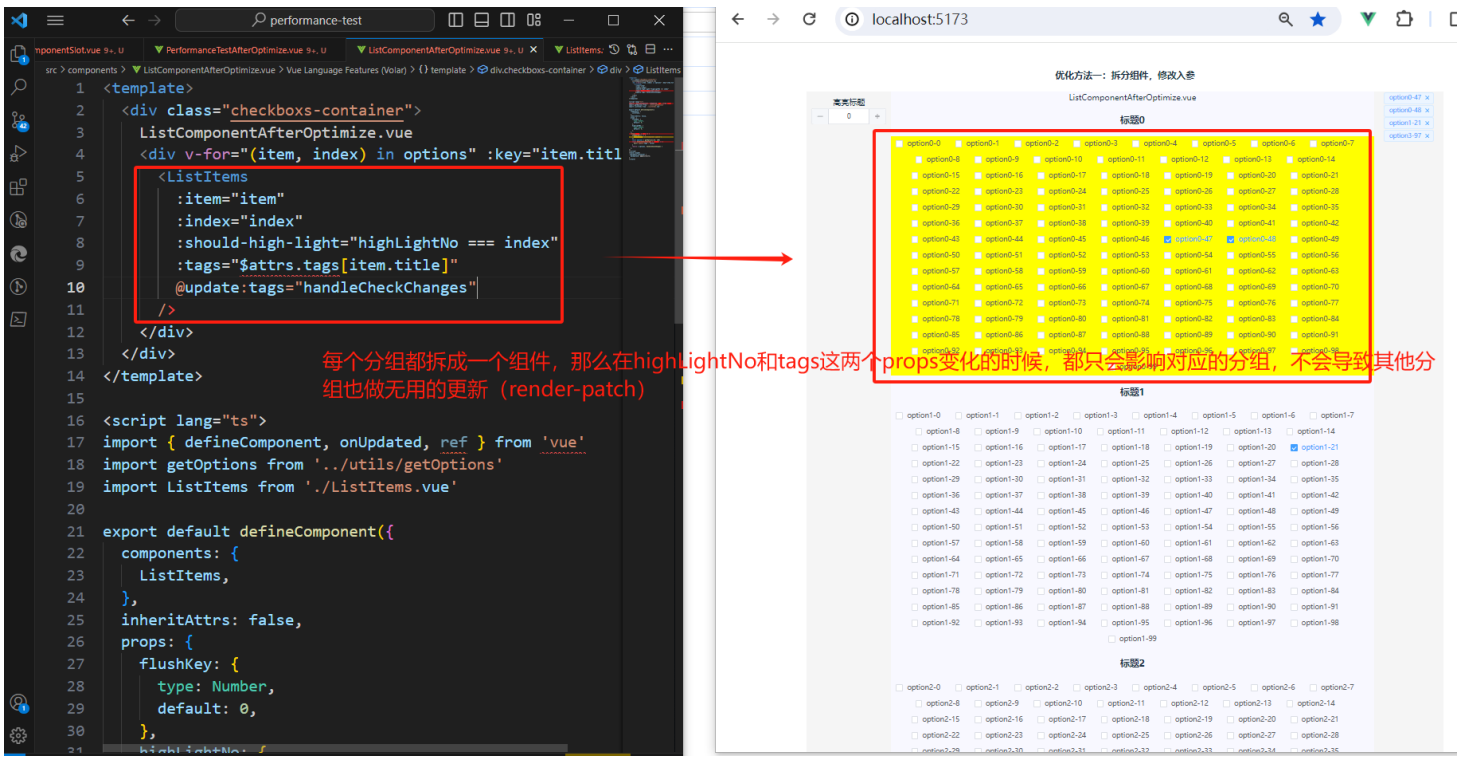
## 优化方式2

```

4 import 'element-plus/theme-chalk/index.css'
5
6 // 未优化前
7 // import App from './PerformanceTestBeforeOptimize.vue'
8 // 优化方式1
9 import App from './PerformanceTestAfterOptimize.vue'
10 // 优化方式2
11 // import App from './PerformanceTestAfterOptimize2.vue'
12 // el-checkbox不需要使用slot的情况下，使用slot和非slot的性能差异
13 // import App from './PerformanceTestBeforeOptimizeSlotTest.vue'
14
15 const app = createApp(App)

```

1. 首先，重新切割了ListComponent这个组件的颗粒度



## 2. 需要对tags的数据结构修改

把它变成key-value的对象，能以标题作为key，找到对应的value（选中的checkboxes），实现数据更新也尽量精确到对应的分组，比如我选中和取消选中的tags是属于标题3的，那只有标题3对应的ListItems这个组件需要更新，其他ListItems不需要

```
performance-test
Welcome | package.json | TS main.ts | PerformanceTestAfterOptimize2.vue | ListComponentAfterOptimize2.vue | ListComponent.vue | ListComponentSlot.vue | PerformanceTestAfterOptimize.vue
src > PerformanceTestAfterOptimize.vue > Vetur > {} "PerformanceTestAfterOptimize.vue" > template > div.flex-container > div.tags-display
9      <ListComponent
10        :tags="tags"
11        :high-light-no="highLightNo"
12        @update:tags="handleTagsChange"
13      />
14
15      <div class="tags-display">
16        <el-tag
17          v-for="tag in tagsArray"
18          :key="tag"
19          class="mx-1"
20          closable
21          @close="handleClose(tag)"
22        >
23          {{ tag }}
24        </el-tag>
25      </div>
26    </div>
27  </template>
28
29  <script lang="ts">
30    import { computed, defineComponent, ref } from 'vue'
31    import ListComponent from './components/ListComponentAfterOptimize.vue'
32    export default defineComponent({
33      components: {
34        ListComponent,
35      },
36      setup() {const highLightNo: Ref<number>
37        const highLightNo = ref(0)
38        const tags = ref({})
39        const handleClose = (tag: string) => {
40          Object.values(tags.value).forEach((array) => {
41            const _index = array?.indexOf(tag)
42            const hasEle = _index > -1
43            hasEle && array?.splice( index, 1)
```

```
performanceTestAfterOptimize2.vue 1, U    ▼ PerformanceTestAfterOptimize.vue 8, U    ▼ ListComponentAfterOptimize.vue 6, U    ▼ ListItems.vue 6, U
ges > performance-test > src > components > ▼ ListItems.vue > Vetur > {} "ListItems.vue" > script > default > setup
1  <template>
2    <h3>{{ item.title }}</h3>
3    <el-checkbox-group
4      :model-value="$attrs.tags"
5      :class="[shouldHighLight ? 'bg-yellow' : '']"
6      @update:model-value="handleCheck"
7    >
8      <el-checkbox v-for="label in item.list" :key="label" :label="label"
9        {{ label }}
10     </el-checkbox>
11   </el-checkbox-group>
12 </template>
13
14 <script lang="ts">
15 import { defineComponent, onUpdated, watch } from 'vue'
16 // import ListItem from './ListItem.vue'
17 export default defineComponent({
18   components: {},
19   props: { ...
32   },
33   setup(props, { attrs, emit }) {
34     onUpdated(() => {
35       console.log(props.item.title, 'ListItems update')
36     })
37     const handleCheck = (v) => {
38       emit('update:tags', { key: props.item.title, value: v })
39     }

```

```
1 <template>
2   <div class="checkboxs-container">
3     ListComponentAfterOptimize.vue
4     <div v-for="(item, index) in options" :key="item.title">
5       <ListItems
6         :item="item"
7         :index="index"
8         :should-high-light="highlightNo === index"
9         :tags="$attrs.tags[item.title]"
10        @update:tags="handleCheckChanges"
11      />
12    </div>
13  </div>
14 </template>
15
16 <script lang="ts">
17 import { defineComponent, onUpdated, ref } from 'vue'
18 import getOptions from '../utils/getOptions'
19 import ListItems from './ListItems.vue'
20
21 export default defineComponent({
22   components: { ...
23 },
24   inheritAttrs: false,
25   props: { ...
26 },
27   setup(props, { emit }) {
28     onUpdated(() => {
29       console.log('ListComponent update')
30     })
31     const options = getOptions(10, 100)
32     const handleCheckChanges = (value) => {
33       emit('update:tags', value)
34     }
35     return { options, handleCheckChanges }
36   },
37 })
38 </script>
```



```
oreOptimize.vue 4, U | ListComponent.vue 3, U | PerformanceTestAfterOptimize2.vue 1, U | PerformanceTestAfterOptimize.vue 8, U X | ListComponentAfterOptimize.vue 6, U | ListItems
packages > performance-test > src > PerformanceTestAfterOptimize.vue > Vue Language Features (Volar) > {} template > div.flex-container > ListComponent
5 |     高亮标题
6 |     <el-input-number v-model="highLightNo" />
7 | </div>
8 |
9 | <ListComponent
10 |   :tags="tags"
11 |   :high-light-no="highLightNo"
12 |   @update:tags="handleTagsChange"
13 | />
14 |
15 | <div class="tags-display">...
25 | </div>
26 | </div>
27 | </template>
28 |
29 | <script lang="ts">
30 | import { computed, defineComponent, ref } from 'vue'
31 | import ListComponent from './components/ListComponentAfterOptimize.vue'
32 | export default defineComponent({
33 |   components: {
34 |     ListComponent,
35 |   },
36 |   setup() {
37 |     const highLightNo = ref(0)
38 |     const tags = ref({})
39 |     const handleClose = (tag: string) => { ...
45 |   }
46 |   const tagsArray = computed(() => Object.values(tags.value).flat())
47 |   const handleTagsChange = (v) => {
48 |     // tags.value = Object.assign(tags.value, v)
49 |     const { key, value } = v || {}
50 |     !!key && (tags.value[key] = value)
51 |     console.log(tags.value)
52 |   }
53 |   return { highLightNo, tags, handleClose, tagsArray, handleTagsChange }
54 | },
```

仅对tags这个对象对  
对应key的属性做修改

### 3. highLightNo转换一下再给到ListItems

- 不转换的话，每次都会把新的0, 1, 2传给所有的ListItems，会导致所有ListItems更新
- 转化后，只有highLightNo等于当前分组index，也就是高亮改变的ListItems才需要更新，其他不需要

```

<ListItem
  :item="item"
  :index="index"
  :should-high-light="highlightNo === index"
  :tags="$attrs.tags[item.title]"
  @update:tags="handleCheckChanges"
/>
</div>
</div>
</template>

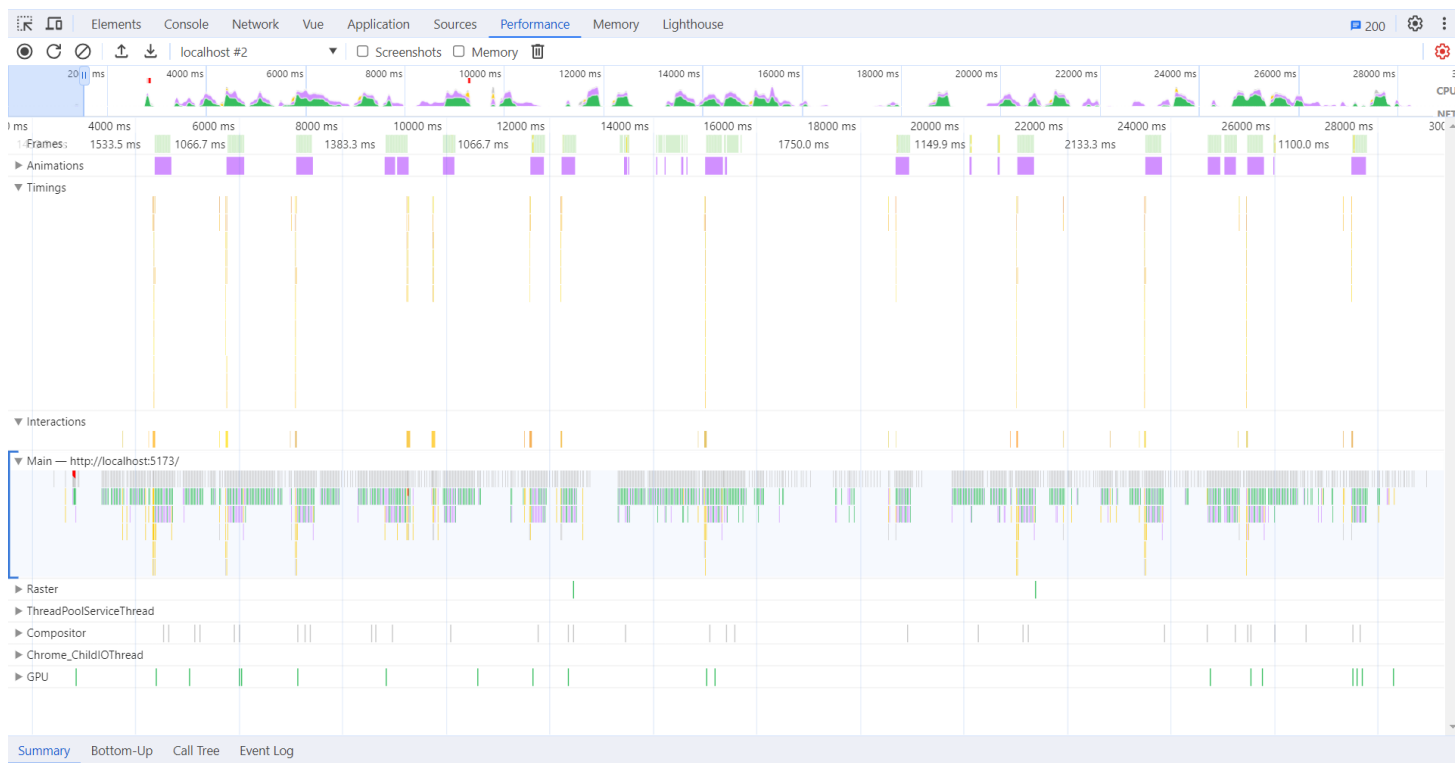
<script lang="ts">
import { defineComponent, onUpdated, ref } from 'vue'
import getOptions from '../utils/getOptions'
import ListItem from './ListItem.vue'

export default defineComponent({
  components: {
    ListItem,
  },
  inheritAttrs: false,
  props: {
    flushKey: {
      type: Number,
      default: 0,
    },
    highlightNo: {
      type: Number,
      default: 0,
    },
  },
  setup(props, { emit }) {
    onUpdated(() => {
      console.log('ListComponent update')
    })
  }
})

```

这样下来，每次操作高亮分组，点击checkbox，通过tags删除选中checkbox的操作使，都只会影响对应的分组，大大减少了ListItems的更新，从而影响checkbox的更新

最后效果如下



## 优化方式3

看看第二种方式

其实第二种方式和第一种大致是一样的，核心逻辑也是一样的，但是考虑到修改数据结构，例如上面第一种方法提到的，将tags从原来的一维数组换成keyValue对象，可能会引起较大的逻辑变动，无法把握可靠性，那就可以使用v-memo的方法

请看main.ts引用的这个文件

```
10 // 优化方式2
11 import App from './PerformanceTestAfterOptimize2.vue'
12 // el-checkbox不需要使用slot的情况下，使用slot和非slot的性能差异
```

tags并没有像上面一样被改成了对象，依然保留一维数组

```

1 <template>
2 <h3>优化方法2: 在-的基础上, 入参修改量较少, 使用v-memo</h3>
3 <div class="flex-container">
4   <div class="highlight-index">
5     高亮标题
6     <el-input-number v-model="highLightNo" />
7   </div>
8   <ListComponent v-model:tags="tags" :high-light-no="highLightNo" />
9   <div class="tags-display">
10    <el-tag
11      v-for="tag in tags"
12      :key="tag"
13      class="mx-1"
14      closable
15      @close="handleClose(tag)"
16    >
17      {{ tag }}
18    </el-tag>
19  </div>
20 </div>
21 </template>
22
23 <script lang="ts">
24 import { defineComponent, ref } from 'vue'
25 import ListComponent from './components/ListComponentAfterOptimize2.vue'
26
27 export default defineComponent({
28   components: {
29     ListComponent,
30   },
31   setup() {
32     const highLightNo = ref(0)
33     const tags = ref([])
34     const handleClose = (tag: string) => {
35       tags.value.splice(tags.value.indexOf(tag), 1)
36     }
37     return { highLightNo, tags, handleClose }
38   },
39 })

```

只是他的子组件ListItems里面使用了**v-memo**来判断每一个checkbox是否需要更新，如果两次的选中情况一致，则代表不需要更新

```
▼ ListItem2.vue 9+, U X  ▼ ListComponent.vue 8, U  ▼ ListComponentSlot.vue 9+, U  ▼ PerformanceTestAfterOptimize.vue 9+, U  ▼ ListComponentAfterOptimize.vue 9+, U
src > components > ▼ ListItem2.vue > Vetur > {} "ListItem2.vue" > template > el-checkbox-group
1  <template>
2    <h3>{{ item.title }}</h3>
3    <el-checkbox-group
4      v-memo="[shouldUpdate, shouldHighLight]"
5      :model-value="$props.tags"
6      :class="[shouldHighLight ? 'bg-yellow' : '']"
7      @update:model-value="$attrs['onUpdate:tags']"
8    >
9      <el-checkbox v-for="label in item.list" :key="label" :label="label">
10        {{ label }}
11      </el-checkbox>
12    </el-checkbox-group>
13  </template>
14
15  <script lang="ts">
16    import { defineComponent, onUpdated, ref, watch } from 'vue'
17    import _ from 'lodash-es'
18    // import ListItem from './ListItem.vue'
19    export default defineComponent({
20      components: {},
21      props: { ...
38    },
39    setup(props, { attrs }) {
40      const shouldUpdate = ref()
41      onUpdated(() => { ...
44    watch(
45      () => props.tags,
46      (newValue, oldValue) => {
47        const diffValues = _.difference(newValue, oldValue)
48        shouldUpdate.value = props.item.list.includes(diffValues[0] || '')
49        ? !shouldUpdate.value
50        : shouldUpdate.value
51    }
```

!!!!

v-memo的更新判断逻辑是由开发者根据实际情况来制定，例子中只是实现基本的演示功能，伪代码的v-memo判断是有bug的，但其实在这次的业务优化中我们最终也是选择了v-memo

最终效果

