廣州新華學院 GUANGZHOU XINHUA UNIVERSITY **前端开发框架与应用实验报告**

警示

1.实验报告如有雷同，雷同各方当次实验成绩均以 0 分计。
2.在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计。
3.实验报告文件以 PDF 格式提交。

标签页实验

【实训链接】

https://glpla.github.io/vue/index.html

【实验题目】

标签页。

【实验目的】

1. 掌握异步组件的使用。

2. 掌握动态组件的使用。

3. 掌握缓存组件的作用和使用。

4. 掌握组件样式使用时的基本原则和考虑。

5. 进一步熟悉组件生命周期函数的执行。

【实验内容】

1. 创建标签页项目。

2. 编辑标签页组件和各测试组件。

3. 设计标签页样式。

4. 普通方式引入组件，当标签页切换时，查看各组件的生命周期函数和网络请求情况。

5. 为组件应用缓存，查看各组件的生命周期函数。

6. 异步方式引入组件，查看网络请求情况。

【前置需求】

1. 操作系统 Window 10+ 基本使用；

2. 互联网基本使用；

3. 办公文档基本使用；

4. 编辑器 Vs Code 的基本使用；

5. 浏览器 Chrome 的基本使用；

【开发环境】

1. 操作系统 Window 10+；

2. 文本编辑器 Vs Code；

3. 谷歌浏览器 Chrome；

4. 截图 Snipaste；

5. 服务器端 Node.js；

【实验要求】

1. 规范开发；独立完成；突出个人设计特点和风格；

2. 实验报告：采用学院统一下发的实验报告格式文件，以文字说明，配以必要的效果图片或核心代码，展示并说明数据来源、实施过程、各部分功能、具体内容和实现细节；

3. 实验报告导出为 PDF，按照要求命名并提交到学习通；未在规定时间内按要求提交，视为无效作业，不得分；

4. 格式规范，请参考 https://glpla.github.io/course/paper.html ；

【实验过程记录】

1. 创建基于 vite 的 Vue3 项目 vite-tabs，并使用 Vs Code 打开。

・不使用路由

・不使用状态管理

・不采用格式检查

2. 修改入口组件 App.vue 为标签页主页；删除所有内容，仅保留主框架。

```
<script setup>

</script>

<template>

</template>

<style scoped>

</style>
```

3. 创建各测试组件，包括加载和卸载生命周期函数。每个组件使用唯一根节点并指定同名类。

・Home.vue

```
<script setup>
import { onMounted, onUnmounted } from 'vue'
onMounted(() => {
  console.log('home mounted');
})
onUnmounted(() => {
  console.log('home unmounted');
})
</script>

<template>
  <div class="home">home</div>
</template>

<style scoped></style>
```

・Info.vue

```
<script setup>
```

```
import { onMounted, onUnmounted } from 'vue'
onMounted(() => {
  console.log('info mounted');
})
onUnmounted(() => {
  console.log('info unmounted');
})
</script>

<template>
  <div class="info">info</div>
</template>

<style scoped></style>
```

・Work.vue

```
<script setup>
import { onMounted, onUnmounted } from 'vue'
onMounted(() => {
  console.log('work mounted');
})
onUnmounted(() => {
  console.log('work unmounted');
})
</script>

<template>
  <div class="work">work</div>
</template>

<style scoped></style>
```

・Team.vue

```
<script setup>
import { onMounted, onUnmounted } from 'vue'
onMounted(() => {
  console.log('team mounted');
})
onUnmounted(() => {
  console.log('team unmounted');
```

```
})
</script>

<template>
  <div class="team">team</div>
</template>

<style scoped></style>
```

4. 在标签页主页 App.vue 逻辑<script>中，使用**普通方式**引入各测试组件及对应的包。

```
import { ref, shallowRef } from 'vue';
import Home from './tabs/Home.vue'
import Info from './tabs/Info.vue'
import Team from './tabs/Team.vue'
import Work from './tabs/Work.vue'
```

5. 在标签页主页 App.vue 逻辑<script>中定义响应式变量和方法。其中，变量 ind 用来表示动态渲染组件的索引；变量 list 为动态组件列表；方法 setInd 用来改变 ind 从而实现组件切换，达到动态渲染组件的目的。

```
let ind = ref(0)
const list = shallowRef([{
  tag: 'Home',
  component: Home
}, {
  tag: 'Info',
  component: Info
}, {
  tag: 'Team',
  component: Team
}, {
  tag: 'Work',
  component: Work
}])
const setInd = (index) => {
  ind.value = index
}
```

6. 在标签页主页 App.vue 结构< template >中使用列表渲染渲染按钮并绑定 setInd 事件。

```
<div class="tabs">
```

```
    <div class="btns">
      <button class="btn" v-for="(item, index) in list"
@click="setInd(index)">{{ item.tag }}</button>
    </div>
    <KeepAlive>
      <component :is="list[ind].component"></component>
    </KeepAlive>
  </div>
```

7. 在标签页主页 App.vue 样式< style>中设计样式。采用弹性盒子布局：整体垂直方向布局；交互按钮水平布局；均采用 gap 分隔。

```
  .tabs {
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 10px;
    height: 100vh;
  }

  .btns {
    display: flex;
    justify-content: center;
    gap: 10px;
    width: 100%;
    padding: 10px 0;
    border-bottom: 1px solid #ccc;
  }

  .btn {
    padding: 4px 10px;
    border-radius: 4px;
  }
```

8. 运行项目，单击页面按钮，查看页面各组件的渲染情况；同时，按 F12 打开浏览器开发者视图，切换至控制台 Console。当组件切换时，观察控制台组件的输出。可以看出：仅仅当渲染某个组件时，该组件才挂载；切换到其它组件时，当前组件卸载。组件频繁的加载和卸载，将影响页面的渲染性能，同时组件的状态如其中的数据，将丢失。项目执行时，控制台输出如图 1 所示。

```
  npm run dev
```

图 1 组件切换时的加载和卸载情况

9. 在开发者视图，选择网络 Network。按 F5 刷新页面或在浏览器任意位置，右键 → 刷新。可以看出：当前网络加载或请求为 19 次。当组件切换时，网络请求没有变化，已经全部加载完毕。如图 2 所示。具体情况，视项目的内容和组件的设计而定。

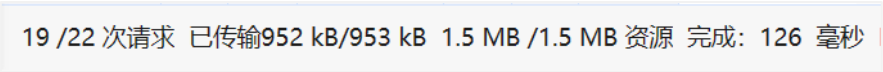

图 2 组件切换时，网络请求不变

10. 在标签页主页 App.vue 结构< template >中**添加**缓存组件<KeepAlife>，包裹节点组件<component>。返回浏览器，单击各按钮，再次观察各组件的渲染情况和加载情况。可以看出：使用缓存组件后，组件只有加载，没有卸载。当组件再次被切换回来时，可以快速显示，提高了页面渲染性能。项目执行时，控制台输出如图 3 所示。

```
<div class="tabs">
  <div>
    <button v-for="(item, index) in list" @click="setInd(index)">{{ item.tag }}</button>
  </div>
  <KeepAlive>
    <component :is="list[ind].component"></component>
  </KeepAlive>
</div>
```

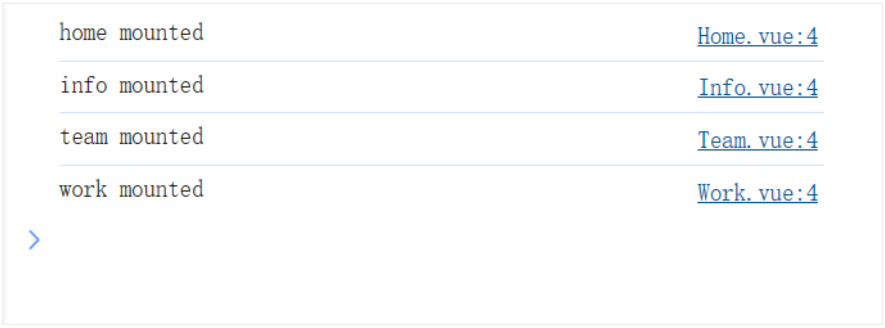

图 3 使用缓存时，各组件切换时的渲染和加载情况

11. **修改**标签页主页 App.vue 逻辑<script>中组件的引入方式为**异步引入**。仅仅当该组件挂载时，才被引入。组件多的情况下，可以显著缩短首屏加载时间。

```
import { defineAsyncComponent, ref, shallowRef } from 'vue';
const Home = defineAsyncComponent(() => import('./tabs/Home.vue'));
const Info = defineAsyncComponent(() => import('./tabs/Info.vue'));
const Team = defineAsyncComponent(() => import('./tabs/Team.vue'));
const Work = defineAsyncComponent(() => import('./tabs/Work.vue'));
```

12. 在开发者视图，选择网络 Network。按 F5 刷新页面或在浏览器任意位置，右键 → 刷新。当前网络加载或请求为 16 次。单击各按钮，动态渲染组件，可以看出：网络请求次数逐渐增加。当所有的组件都加载完毕后，不再触发新的网络请求。如图 4 所示。具体情况，视项目的内容和组件的设计而定。

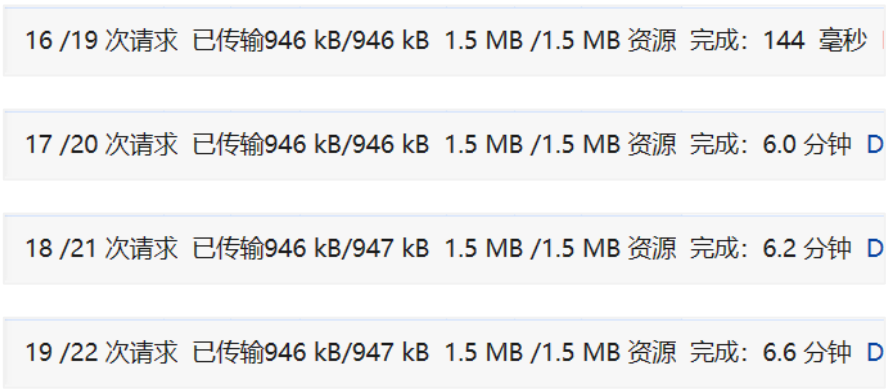

图 4 页面刷新后的网络请求次数变化

13. 为按钮添加动态样式。当前按钮选中时候，指定 active 样式。如图 5 所示。

```
.btn.active {
  background-color: #f40;
  color: #fff;
```

```
}
```

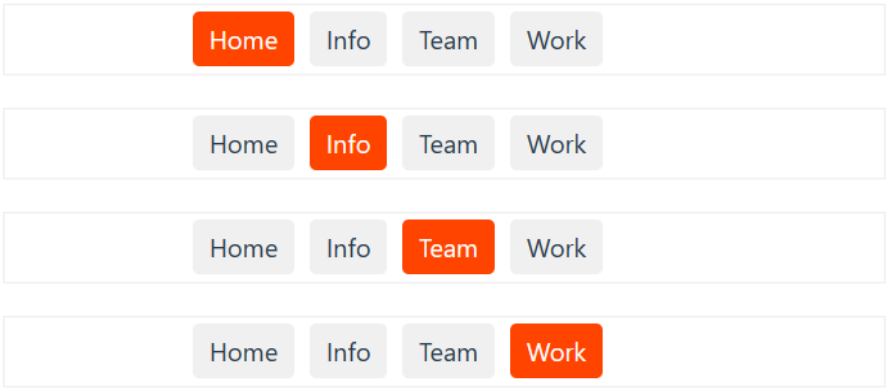

图 5 标签页当前按钮活动样式

【参考效果和参考代码】

略。

【实验总结】

通过实验，掌握了异步组件和缓存组件的使用；了解到组件的不同运用，对系统的性能体现有着决定性的影响；进一步熟悉了项目的开发流程、调试和检查方法；对弹性盒子的使用有了更深的理解，为后期综合项目的开发和实战积累了经验。

开发过程中，为了深刻理解实验内容和要求，组件仅使用普通文字代替。后续学习中，将把相关知识点和技能运用到具体的实战项目，学以致用。

1. 异步组件的按需加载可以提高页面渲染速度。

2. 缓存组件可以保留组件在 DOM 树中，需要时，可以快速激活。

3. 组件使用唯一根节点并指定同名类，有利于父组件布局使用。

【拓展思考】

1. 如何使用字体图标美化 UI?

2. 如何使用动画改善用户体验?