Vue DAY05

父子组件搭建页面的基本结构

Father.vue

```vue
<template>
  <div>
    <!-- views/Father.vue 父组件-->
    <h1>测试示例 - 自定义组件</h1>
    <counter></counter>
    <counter></counter>
  </div>
</template>
<script>
import Counter from '../components/Counter.vue'
export default {
    components: {
        Counter: Counter
    }
};
</script>
```

Counter.vue

```vue
<template>
  <div>
    <button @click="n--">-</button>
    {{n}}
    <button @click="n++">+</button>
  </div>
</template>
```

在页面中使用counter标签就相当于将子组件中的内容搬到父组件的相应位置，进行显示。

父组件向子组件传参：

Father.vue

```vue
<template>
    <div>
        <counter></counter>
        <counter />
        <counter :min="0" :max="5" />
    </div>
</template>
```

父组件中传递参数

Counter.vue

```vue
<template>
    <!-- Counter.vue 计数器组件 -->
    <button :disabled="n==min" ....>-</button>
    <span>{{n}}</span>
    <button :disabled="n==max" ....>+</button>
</template>

<script>
    export default {
        props : ['min', 'max']
    }
</script>
```

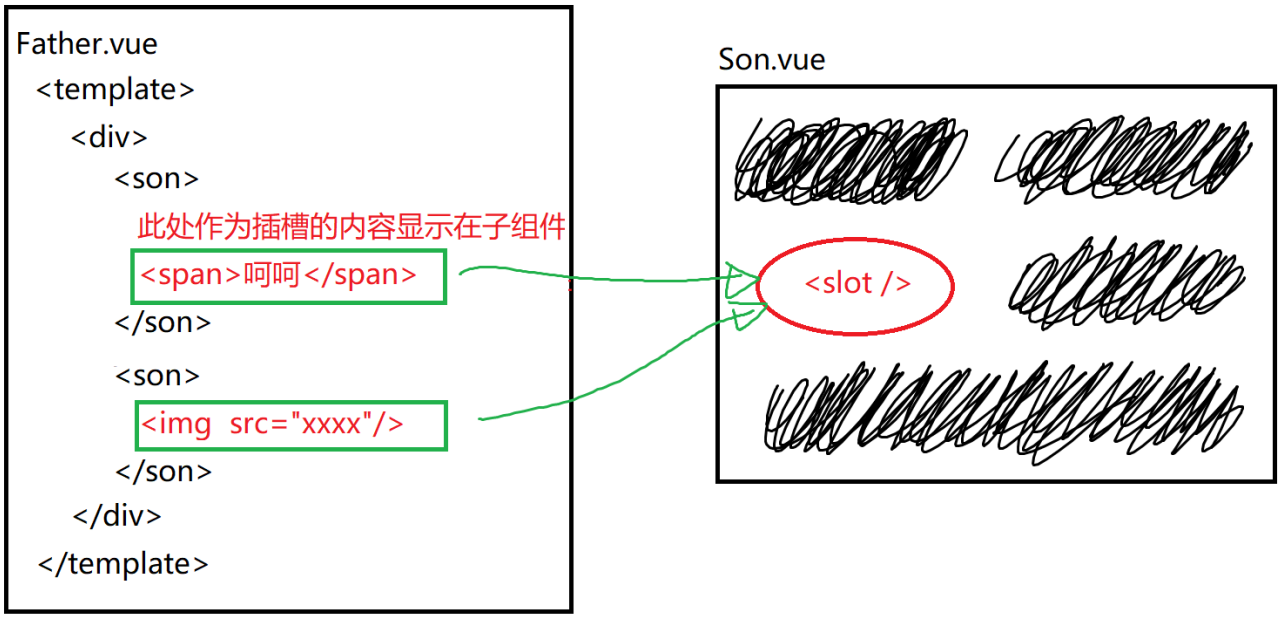
template中使用属性

此处声明属性

组件插槽（slot）

平时在设计子组件时，绝大多数布局都已经在子组件中完成了定义，但是有些布局需要在父组件使用子组件时动态设置，这时就可以使用**组件插槽**来实现。 在设计子组件时，可以再布局中定义一个插槽位置，并给予基础样式，父组件在使用的时候就可以通过`slot`属性来动态设置需要显示的内容。

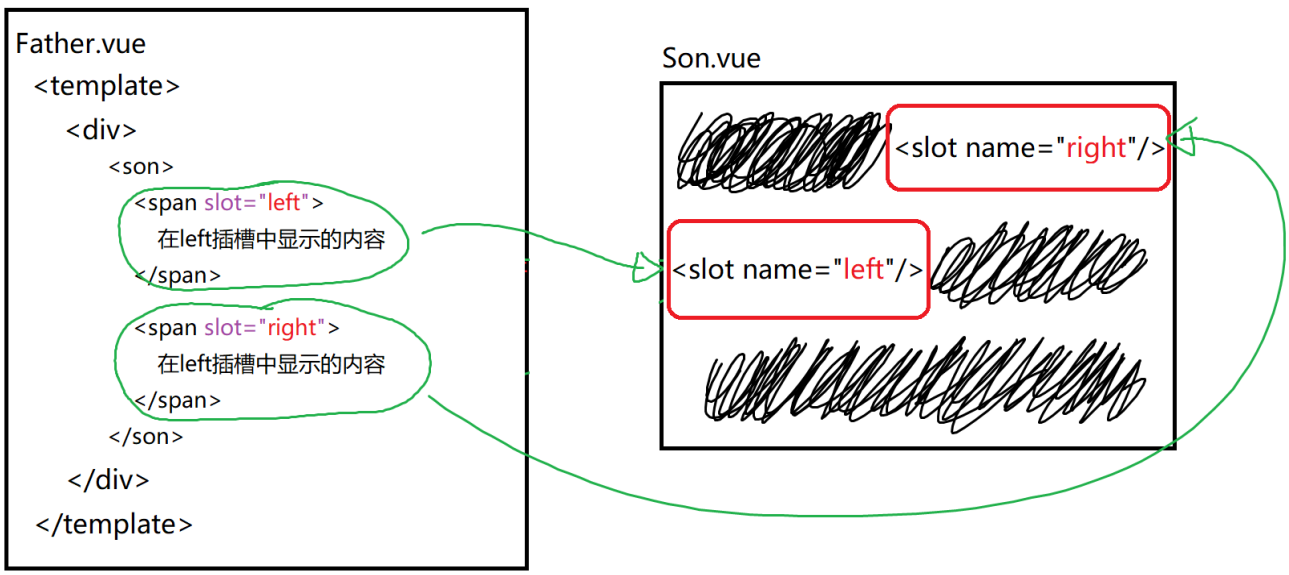
默认插槽



实现步骤：

1. 在子组件中，设计一个插槽位置，通过 `<slot/>` 组件来指定插槽的位置，定义通用的基础布局样式。
2. 在父组件使用子组件时，开始标签与结束标签中的内容将会作为默认插槽内容显示在子组件的 `<slot/>` 的位置。

具名插槽（带有名字的插槽）



`VueCLI` 脚手架中的路由系统 `VueRouter`

一个大型的网站需要由多个页面共同组成，页面之间还要支持相互跳转。

开发方式有两种：

1. 传统方案：每一个页面都是一个独立的`html`文件，通过超链接`<a href="">xx</a>`来实现页面之间的跳转。

   特点：先清空浏览器中的所有内容，然后重新显示页面（浏览器会闪一下）

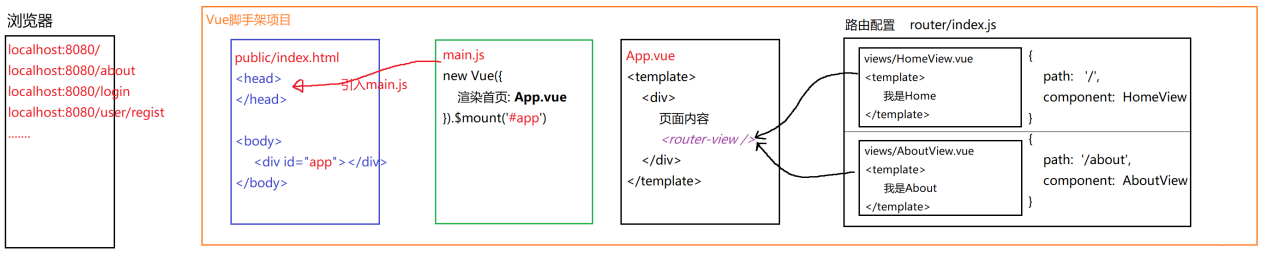
2. 新方案：一个项目中只有一个`html`页面，根据不同的路径，通过`ajax`的方式动态加载页面的局部内容（替换掉`html`中核心的`div.#app`）。

   特点：不需要切换html页面（浏览器不会重新加载新的`dom`树），只需要更新局部内容即可。

`VueCLI`使用的`VueRouter`路由属于**新方案。**

专业一些称`vueCLI`项目为**单页面应用（`Single Page Application`缩写`SPA`）**。意味着`vueCLI`项目无论设计多少个页面，本质上都只有一个`HTML`网页。由`VueRouter`来根据请求路径动态加载`html div#app`里的内容。

流程图：



通过`src/router/index.js`来指定路由地址与组件之间的映射关系：

```
import HomeView from '../views/HomeView.vue'

const routes = [
    {
        path: '/',
        name: 'home',
        component: HomeView
    },
    {
        path: '/register',
        name: 'register',
        component: () => import('../views/Register.vue')
    },
]
```

通过`component`指定组件位置时，有上述两种方式：标准模式、懒加载模式。

标准模式：

```
import HomeView from '../views/HomeView.vue'

{
    path: '/',
    name: 'home',
    component: HomeView
},
```

对于标准模式加载组件，在`import`的时候就已经将组件加载到内存中了。这样的话，当`Vue`项目初始化时，随着路由系统的加载将直接下载引入该组件。（虽然可能不需要立即看到该组件，但是已经下载完毕后，内存中已经存在，如果需要跳转到该组件时，不需要重新下载，直接就可以显示组件内容）

懒加载模式

```
{
    path: '/register',
    name: 'register',
    component: () => import('../views/Register.vue')
},
```

懒加载模式使用箭头函数的方式引入组件，这种方式使得在需要显示该组件时再去下载相应资源，真正做到随用随下载。这种方式不会在项目初始化时就下载大量路由组件，极大的释放项目首页资源数据量的压力，防止出现首页白屏等待现象。

结论：

除了首页这种必须加载的资源使用标准模式外，其余都可以使用懒加载。

基于`VueRouter`路由系统完成页面间的跳转功能

基于组件的方式实现跳转

`<router-link></router-link>`，可以实现路由的跳转：

```
<router-link to="/register">点我去/register</router-link>
<router-link to="/vfor">点我去/vfor</router-link>
<router-link to="/http" replace>点我去/http</router-link>
```

基于代码方式实现跳转（编程式跳转）

```
<button @click="goto">点击后跳转 /vfor</button>
<button @click="goto2">点击后跳转 /login</button>
```

```
goto(){
    this.$router.push('/vfor')
}
goto2(){
    this.$router.push('/login')
}
```

`this.$router`

`this` 表示当前 `vue` 对象，`$router` 是 `vue` 对象中的一个属性。该属性在项目初始化时已经完成赋值。`this.$router` 用于获取当前项目中的路由管理器（`VueRouter` 对象）。 该路由管理器对象中包含了已经配置好的所有的路由对象，每一个路由对象中包含：`path`、`name`、`component` 属性。它还提供了一些方法用于操作路由跳转：

```
this.$router.push('/login')   // 跳转到/login登录页面
this.$router.replace('/login')  // 替换当前路由，到/login登录页面
this.$router.go(-1)  // 回到上一页
.....
```

案例：从电影列表页，跳转到电影详情页。

实现步骤：

1. 准备一个电影详情页面，显示某一部电影的详细信息。
2. 在列表页中点击电影标题后，跳转到详情页。
3. 在详情页中，获取当前选中的电影详情，渲染页面内容。

路由跳转过程中的参数传递问题

上述业务就需要在跳转路由的同时（从 `/http` 跳转到 `/detail`），传递参数（带着选中的电影 `id` 一起跳转）。此处 `vuerouter` 提供了两种参数传递方案。

第一种参数传递的方案 （查询字符串）

使用 `?` 在路径后进行查询字符串拼接：

```
<router-link to="/detail?id=125">去详情</router-link>
```

或

```
this.$router.push('/detail?id=125')
```

在详情页中如下即可获取参数 `id`：

```
let id = this.$route.query.id   // 获取请求路径查询字符串中的参数id的值let
let name = this.$route.query.name  // 获取请求路径查询字符串中的参数name的值
```

`this.$route`

在 `vue` 组件中通过 `this.$route` 可以获取当前组件所对应的路由对象（相当于 `router/index.js` 中定义的一段配置：`{path, name, component}`）。`$router` 是路由管理器，管理了所有的路由对象。

`$router` 提供了路由管理器所拥有的属性和方法，跳转页面等功能。

`$route` 封装当前路由的属性：`name`、`path`、`component`、`query`、`param` 等属性。

第二种参数传递的方案

将参数藏在请求资源路径后，进行参数传递：

```
<router-link to="/detail/125">去详情</router-link>
```

或

```
this.$router.push('/detail/125')
```

Vue 组件的生命周期

整个 vue 项目都是由一个个组件组成的，每个组件各司其职。当需要看到某一个组件时， vue 框架就会创建该组件对象，将该组件的内容挂载到页面上进行显示。当需要跳转页面时， vue 将会从 DOM 树中卸载前一个组件，销毁它；然后创建新的组件，并且将新组件挂载到页面的 DOM 树中显示。**这整个过程都是由 vue 框架自己管理。**

如果遇到了如下需求，该怎么做？

```
当组件对象创建成功后，打印一句话，发送一个请求....
当组件成功挂载到DOM树之后，发送一个请求，加载数据....
当跳转到下一页之前，当前组件销毁之前，打印一句话，释放已经申请的内存资源....
当组件内容由于data中数据的变化，自动更新时......
当.......
当.......
当.......
```

由于 vue 框架管理了组件对象从创建、挂载、使用、卸载整个生命周期，所以遇到了如上需求，**想写代码，都没地方写。**

vue 框架中**组件的生命周期钩子方法**就是为了解决这些问题而设计实现的。这些生命周期钩子方法会再使用组件的过程中的响应时间节点自动被调用。所以，**如果有些业务需求需要在：组件创建、组件挂载、组件使用、组件卸载等时间点执行时，就可以在相应的生命周期钩子方法内进行编写。**

常见的生命周期钩子方法如下：

```
export default {
    beforeCreate(){},  // 组件对象创建之前自动调用
    created(){},  // 组件对象创建完毕后自动调用
    beforeMount(){},   // 组件对象挂载之前自动调用
    mounted(){},   // 组件对象挂载到dom之后自动调用
    beforeUpdate(){},   // 组件对象数据更新之前自动调用
    updated(){},   // 组件对象数据更新完毕后自动调用
    beforeDestroy(){},   // 组件对象销毁之前自动调用
    desctroyed(){}   // 组件对象销毁之后自动调用
}
```

Vue组件库

在开发vue项目的过程中，为了组件复用，大厂们都会将一些常见的页面结构封装为组件，设计一些强大的功能。UI美观。这些组件最终组成了一整套组件库。开源，造福广大程序员。

1. 饿了么团队开发了一套基于PC端的vue组件库。ElementUI。

```
vue2版本:
https://element.eleme.cn/#/zh-CN/component/installation
```

```
vue3版本:  Element-Plus
https://element-plus.gitee.io/zh-CN/component/button.html
```

2. 有赞团队开发的一套基于移动端的vue组件库：vant。

```
vue2 版本:
https://vant-contrib.gitee.io/vant/v2/#/zh-CN/

vue3 版本:
https://vant-contrib.gitee.io/vant/#/zh-CN
```

经典错误集合

```
▸[Vue warn]: Property or method "clickMe" is not defined on the instance but referenced during render. Make sure that this property is reactive, either in the data option, or for class-based components, by initializing the property. See: https://v2.vuejs.org/v2/guide/reactivity.html#Declaring-Reactive-Properties.

(found in <Root>)                                                    vue.js:5108
▸[Vue warn]: Invalid handler for event "click": got undefined

(found in <Root>)                                                    vue.js:5108
```

属性或方法`clickMe`并没有在vue对象中定义。要么在data段中声明，要么定义成组件的成员。

`Module not found: Error: Can't resolve './views/Basic.vue' in 'D:\code2209\01_Vue\day02\demo\vue-project\src\router`

模块找不到错误。好好检查代码中编写的这个路径是否可以找到对应的组件。当前这个错误就是找不到这个组件（`'./views/Basic.vue'`）而出的错误。