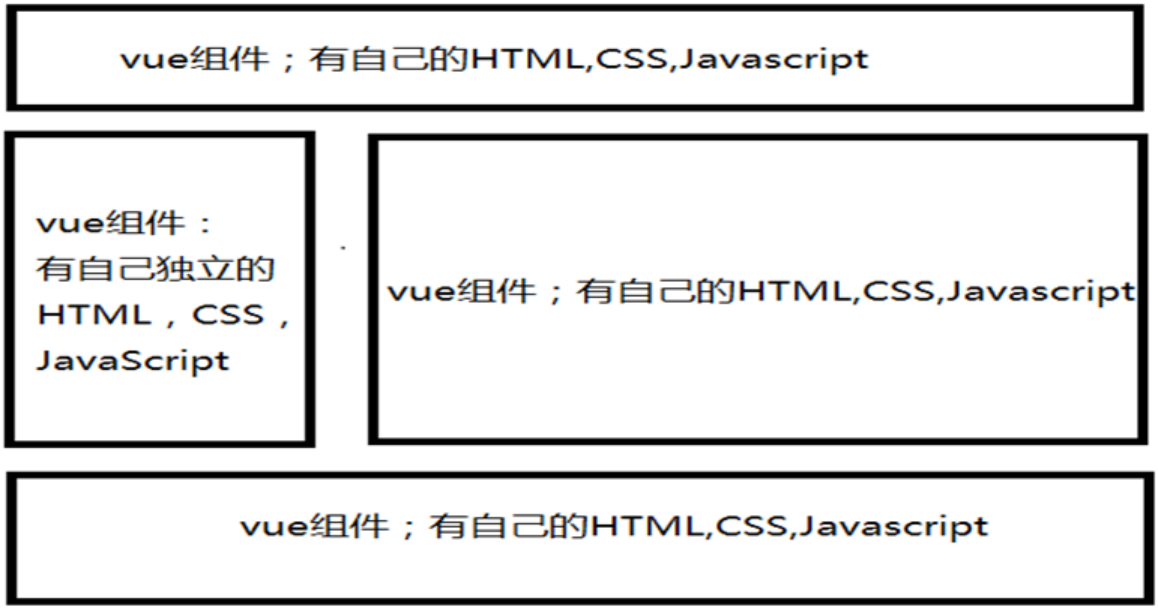
1.Vue组件的定义、注册方式和模板使用

1.1 组件的定义

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码.

vue组件是把页面（html代码，CSS代码）进行模块化

如下图所示，一个页面分四个部分，每个部分是个vue组件，而页面成了vue组件的容器。



vue.js中创建组件有三个步骤：定义组件，注册组件以及使用组件

定义组件

- 当使用构建步骤时，我们一般会将 Vue 组件定义在一个单独的 `.vue` 文件中，这被叫做单文件组件 (简称 SFC)：

定义组件名的方式有两种：

- 使用 kebab-case：使用 kebab-case (短横线分隔命名) 定义一个组件（kebab发音/kI'ba:b/）
- 使用 PascalCase：(驼峰式命名) 定义一个组件

当使用 kebab-case (短横线分隔命名) 定义一个组件时，你也必须在引用这个自定义元素时使用 kebab-case，例如 `<my-component-name>`。当使用 PascalCase (驼峰式命名) 定义一个组件时，你在引用这个自定义元素时两种命名法都可以使用。也就是说 `<my-component-name>` 和 `<MyComponentName>` 都是可接受的。

1.2 组件的分类

- 全局组件
- 局部组件

  通过选项components定义，但只能在当前Vue实例中使用

1.3 引用模版

将组件内容放到`<template>`中引用。

代码示例

组件：GlobalComponentA.vue文件

```
<!--template表示模板，页面html元素写在template标签内-->
<template>
    <h3>{{info}}</h3>
    <a href='http://www.baidu.com'>百度</a>
</template>

<!--js代码写在script标签中-->
<script>
    //export default 默认导出组件
    export default {
        //组件名称
        name: "GlobalComponentA",
        setup() {
            return {
                info: "全局组件"
            }
        }
    }
</script>

<!--css样式写在style标签内-->
<style scoped>

</style>
```

组件：HelloWorld.vue文件

```
<template>
    <h2>{{message}}</h2>
    <!--全局组件-->
    <GlobalComponentA></GlobalComponentA>
</template>

<script>
    export default {
        name: 'HelloWorld',
        setup() {
            return {
                message: "HelloWord"
            }
        }
    }
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>

</style>
```

HomeView.vue文件

```vue
<template>
    <div class="home">
        <h1>{{message}}</h1>
        <!--使用短中线名称的组件-->
        <global-component-a></global-component-a>
        <h2>{{msg}}</h2>
        <!--局部组件-->
        <HelloWorld></HelloWorld>
    </div>
</template>

<!--<script setup> 语法糖里面的代码会被编译成组件setup()函数的内容，
不需要通过return暴露声明的变量、函数以及import引入的内容，
即可在<template/>使用，并且不需要些export default{}-->
<script setup>
    import {ref} from 'vue'
    import HelloWorld from '@/components/HelloWorld.vue'

    // export default {
    //     name: 'HomeView',
    //     setup() {
    //         return {
    //             message: "HomeView"
    //         }
    //     },
    //     //局部组件
    //     components: {
    //         HelloWorld
    //     }
    // }
    const message = ref("HomeView")
    const msg = ref("局部组件")
</script>
```

2. 组件间数据的通信

2.1 父子组件

在一个组件内部定义另一个组件，称为父子组件

子组件只能在父组件内部使用

默认情况下，**子组件无法访问父组件中的数据**，每个组件实例的作用域是独立的

2.2 组件间数据通信

2.2.1 子组件访问父组件的数据（父传子）

- 在调用子组件时，绑定想要获取的父组件中的数据在子组件内部，使用**props**选项声明获取的数据，即接收来自父组件的数据。
- 即父组件通过props向下传递数据给子组件。
- 注：组件中的数据存在方式共有三种：变量或者常量、props、computed

prop命名，可以在在组件中使用`postTitle`（驼峰名称），在html中是使用短横线命名`post-title`，如下代码示例

```
<template>
    <h3>{{ postTitle }}</h3>
</template>
<script setup>
    import {defineProps} from 'vue'
    defineProps(['postTitle'])
</script>
或者
<script>
    export default {
        props: ['postTitle'],
        setup(props) {
            //setup() 接收 props 作为第一个参数
            console.log(props.postTitle)
        }
    }
</script>

<!-- 其他组件中使用子组件属性（短横线） -->
<blog-post post-title="hello!"></blog-post>
```

父组件传子组件的代码实例

子组件ComponentA.vue

```
<template>
    <h3>子组件内部数据（child-message）：{{childMessage}}</h3>
    <h4>父组件传入值{{parentMessage}}</h4>
</template>

<script setup>
    import {ref, defineProps} from 'vue'

    const childMessage = ref("我是子组件的message属性值")
    defineProps(['parentMessage'])
</script>

<style scoped>

</style>
```

父组件使用子组件并传值

```
<template>
    <div class="home">
        <h1>{{message}}</h1>
        <!--使用子组件ComponentA并传值-->
        <ComponentA :parent-message="parentMessage"></ComponentA>
    </div>
</template>
```

```
<!--<script setup> 语法糖里面的代码会被编译成组件setup()函数的内容，
不需要通过return暴露声明的变量、函数以及import引入的内容，
即可在<template/>使用，并且不需要些export default{}-->
<script setup>
    import {ref} from 'vue'
    import ComponentA from '@/components/ComponentA'

    const message = ref("HomeView")
    const parentMessage = ref('父组件的parentMessage属性值');
</script>
```

单向数据流

所有的 props 都遵循着单向绑定原则，props 因父组件的更新而变化，自然地将新的状态向下流往子组件，而不会逆向传递。这避免了子组件意外修改父组件的状态的情况，不然应用的数据流将很容易变得混乱而难以理解。另外，每次父组件更新后，所有的子组件中的 props 都会被更新到最新值，这意味着你不应该在子组件中去更改一个 prop。若你这么做了，Vue 会在控制台上向你抛出警告。

2.2.2 父组件访问子组件的数据（子传父）

- 第一步：在子组件中使用 $emit(事件名,数据) 触发一个自定义事件，事件名自定义。
- 第二步：父组件在使用子组件的地方监听子组件触发的事件，并在父组件中定义方法，用来获取数据。
- 总结：子组件通过events（事件）给父组件发送消息，实际上就是子组件把自己的数据发送到父组件

子组件ComponentA.vue

```
<template>
    <h3>子组件内部数据（child-message）：{{childMessage}}</h3>
    <input type="button" v-for="item in categories" :value="item.name"
@click="btnClick(item)">
</template>

<script setup>
    import {ref, defineEmits} from 'vue'

    const childMessage = ref("我是子组件的message属性值")
    //测试数据
    const categories = [
        {id: 1, name: "家电"},
        {id: 2, name: "玩具"},
        {id: 3, name: "数码"},
        {id: 4, name: "服装"}
    ]
    //defineEmits自定义事件itemClick
    const emit = defineEmits(['itemClick'])
    //按钮click回调函数
    const btnClick = (item) => {
        //触发自定义事件itemClick
        emit("itemClick", item)
    }
</script>
```

```
<style scoped>

</style>
```

父组件

```
<template>
    <div class="home">
        <h1>{{message}}</h1>
        <!--使用子组件ComponentA并传值-->
        <!--事件名称使用短中线-->
        <ComponentA @item-click="callback"></ComponentA>
    </div>
</template>

<!--<script setup> 语法糖里面的代码会被编译成组件setup()函数的内容，
不需要通过return暴露声明的变量、函数以及import引入的内容，
即可在<template/>使用，并且不需要些export default{}-->
<script setup>
    import {ref} from 'vue'
    import ComponentA from '@/components/ComponentA'

    const message = ref("HomeView")
    const callback=(data)=>{
        console.log(data);
    }
</script>
```

2.2.4 总结组件之间的通讯

3.路由

3.1 定义

- 路由，其实就是指向的意思，当我点击页面上的home按钮时，页面中就要显示home的内容，如果点击页面上的about 按钮，页面中就要显示about 的内容。

3.2 分类

3.2.1 后端路由

例如，分配一个站点，服务器地址是：`http://192.168.1.200:8899`，在这个网站中提供了三个界面

```
http://192.168.1.200:8899/index.html           主页
http://192.168.1.200:8899/about/about.html   关于我们页面
http://192.168.1.200:8899/feedback.html        反馈界面
```

当我们在浏览器输入 `http://192.168.1.200:8899/index.html` 来访问界面的时候，web 服务器就会接收到这个请求，然后把 index.html 解析出来，并找到相应的 index.html 并展示出来，这就是路由的分发，路由的分发是通过路由功能来完成的

4.2.2 前端路由

1、虽然前端路由和后端路由的实现方式不一样，但是原理都有是相同的，其中一个方式

前端路由的功能都是通过 hash 「散列值」和history 来实现的，hash 能兼容低版本的浏览器

2、后端路由每次仿问一个页面都要向浏览器发送请求，然后服务端再响应解析，在这个过程中肯定会存在延迟，但是前端路由中仿问一个新的界面的时候只是浏览器的路径改变了，没有和服务端交互「所以不存在延迟」，这个对用户体验来说是大大的提高。如下所示：

```
http://192.168.1.200:8080/#/index.html
http://192.168.1.200:8080/#/about/about.html
http://192.168.1.200:8080/#/feedback.html
```

由于 web 服务器不会解析 # 后面的东西「所以通过 hash 能提高性能」，但是客户端的 js 可以拿到 # 后面的东西，有一个方法是 window.location.hash 来读取，使用这个方法来匹配到不同的方法上

3、举个例子

```
http://www.xxx.com/path/a/b/c.html?key1=Tiger&key2=Chain&key3=abc#/path/d/e.html
```

我们把这个地址分析一下

```
http：协议
www.xxx.com：域名
/path/a/b/c.html：路由，即服务器上的资源
?key1=Tiger&key2=Chain&key3=abc：这 Get 请求的参数
#/path/d/e.html：hash 也叫散列值，也叫锚点
```

上面的 hash 是和浏览器交互的，其它的都是和服务器进行交互

3.3 Vue 路由

Vue 中的路由，推荐使用官方支持的 vue-router 库

文件Main.vue组件

```
<template>
    <div class="div1">
        <h1>{{title}}</h1>
    </div>
</template>

<script setup>
    import {ref} from 'vue'

    const title = ref("主页")
</script>

<style scoped>
```

```
    .div1 {
        height: 400px;
        width: 400px;
        background-color: green;
    }
</style>
```

组件文件Message.vue

```
<template>
    <div class="div1">
        <h1>{{title}}</h1>
    </div>
</template>

<script setup>
    import {ref} from 'vue'

    const title = ref("消息")
</script>

<style scoped>
    .div1 {
        height: 400px;
        width: 400px;
        background-color: red;
    }
</style>
```

组件文件Mine.vue

```
<template>
    <div class="div1">
        <h1>{{title}}</h1>
    </div>
</template>

<script setup>
    import {ref} from 'vue'

    const title = ref("我的")
</script>

<style scoped>
    .div1 {
        height: 400px;
        width: 400px;
        background-color: yellow;
    }
</style>
```

配置路由规则，在router目录下index.js

```
import {createRouter, createWebHistory} from 'vue-router'
```

```
import HomeView from '../views/HomeView.vue'
import Main from '@/views/Main'
import Message from '@/views/Message'

const routes = [
    //配置路由
    {
        //url请求路径
        path: '/main',
        //组件名称
        name: 'main',
        //组件对象
        component: Main
    },
    {
        path: '/message',
        name: 'message',
        component: Message
    },
    {
        path: '/mine',
        name: 'mine',
        //可以通过import导入已经存在的组件文件
        component: () => import('@/views/Mine')
    }
]

const router = createRouter({
    history: createWebHistory(process.env.BASE_URL),
    routes
})

export default router
```

组件文件App.vue

```
<template>
    <nav>
        <!--router-link 生成超链接a标签-->
        <!-- to 匹配配置路由的path值，显示对应的组件-->
        <router-link to="/main">首页</router-link>
        <router-link to="/message">消息</router-link>
        <router-link to="/mine">我的</router-link>
    </nav>
    <!--路由视图显示当前组件路由跳转的超链接显示内容-->
    <router-view/>
</template>
```

实际应用界面，通常由多层嵌套的组件组合而成，比如，我们 “首页”组件中，还嵌套着 “登录”和 “注册”组件，那么URL对应就是/myHome/login和/myHome/register

vue组件文件:Login.vue

```
<template>
    <h3>{{title}}</h3>
```

```
</template>

<script setup>
    import {ref, onMounted} from 'vue'
    import {useRoute} from 'vue-router'

    const title = ref("我是登录页面")
    const route = useRoute();
    onMounted(() => {
        console.log(route.query.id);
    })
</script>

<style scoped>

</style>
```

vue组件文件:register.vue

```
<template>
    <h3>{{title}}</h3>
</template>

<script setup>
    import {ref, onMounted} from 'vue'
    import {useRoute} from 'vue-router'

    const title = ref("我是注册页面")
    //获取路由对下稿
    const route = useRoute();
    onMounted(() => {
        //获取请求url的params参数
        console.log(route.params.name);
    })
</script>

<style scoped>

</style>
```

组件文件：MyHome.vue

```
<template>
    <h1>{{title}}</h1>
    <!--path：匹配配置路由的path值，显示对应的组件 通过 query 传参数-->
    <router-link :to="{path:'/myHome/login',query:{id:1}}">登录</router-link>
    <router-link :to="{name:'register',params:{name:'zhangsan'}}">注册</router-
link>
    <!--显示子组件-->
    <router-view></router-view>
</template>

<script setup>
    import {ref} from 'vue'
```

```
    const title = ref("我的主页")
</script>

<style scoped>

</style>
```

配置路由规则，在router目录下index.js

```
const routes = [
    //......
    {
        path: '/myHome',
        name: 'myHome',
        //可以通过import导入已经存在的组件文件
        component: () => import('@/views/MyHome'),
        children: [
            {path: '/myHome/login',name:'login',component:
()=>import('@/views/Login')},
            //path路径使用:参数名 占位，在router-link配置中使用params:{'参数名':'参数
值'} 补位
            {path: '/myHome/register/:name',name:'register',component:
()=>import('@/views/Register')}
        ]
    }
    //........
]
```

4.使用axios进行ajax操作

4.1 Axios简介

vue本身不支持发送AJAX请求，需要使用vue-resource、axios等插件实现。

axios是一个基于Promise的HTTP请求客户端，用来发送请求，也是vue3官方推荐的。

参考：GitHub上搜索axios，查看API文档 https://github.com/axios/axios

Axios 是一个基于 Promise 的 HTTP 库，可以用在浏览器和 node.js 中。本质上也是对原生XHR（XmlHttpRequest）的封装,只不过它是Promise 的实现版本，符合新的ES规范。

4.2 Axios基本用法

axios({options})

axios.get(url,{options});

GET传参方式：

1.通过url传参

2.通过params选项传参

POST传参方式：

axios.post(url,data,{options});

默认情况下，axios将JavaScript对象序列化为JSON。要以application / x-www-form-urlencoded格式发送数据，您可以使用以下选项之一。

传参方式：

1.自己拼接为键值对

2.使用transformRequest，在请求发送前将请求数据进行转换

3.如果使用模块化开发，可以使用qs模块进行转换

Axios的请求代码示例

vue项目中执行命令安装axios和qs

```
npm install axios
npm install qs
```

```
<template>
    <h1>{{message}}</h1>
    <input type="button" value="axiox请求1（get）" @click="sendAxiosGet1"/>
    <input type="button" value="axiox请求2（get）" @click="sendAxiosGet2"/>
    <input type="button" value="axiox请求1（post）" @click="sendAxiosPost1"/>
    <input type="button" value="axiox请求2（post）" @click="sendAxiosPost2"/>
    <input type="button" value="axiox请求3（post）" @click="sendAxiosPost3"/>
    <table border="1px" cellspacing="0px">
        <tr>
            <td>编号</td>
            <td>姓名</td>
            <td>年龄</td>
            <td>邮箱</td>
        </tr>
        <hr/>
        <tr v-for="student in students">
            <td>{{student.id}}</td>
            <td>{{student.name}}</td>
            <td>{{student.age}}</td>
            <td>{{student.email}}</td>
        </tr>
    </table>
</template>

<script setup>
    import {reactive, ref} from 'vue'
    import axios from 'axios'
    import qs from 'qs'

    const message = ref("axios请求");
    //数组使用ref，reactive函数包装的数组，修改数组的值不能页面重新刷新
    let students = ref([]);
    const sendAxiosGet1 = () => {
        axios({
```

```
            url: "http://rap2api.taobao.org/app/mock/238982/students",  //请求的服务器url地址
            method: "get",
            params: {p1: "A", p2: "B"} //请求的数据参数
        }).then((response) => {
            //console.log(response)
            //console.log(response.data.students)
            students.value = response.data.students
        }).catch((e) => {
            console.log(e)
        })
    }
    const sendAxiosGet2 = () => {
        axios.get("http://rap2api.taobao.org/app/mock/238982/students", {
            params: {p1: "A", p2: "B"} //请求的数据参数
        }).then((response) => {
            students.value = response.data.students
        }).catch((e) => {
            console.log(e)
        })
    }
    const sendAxiosPost1 = () => {
        //那个返回 204 的是 options 请求，跟 CORS 的跨域支持请求有关
        axios({
            url: "http://rap2api.taobao.org/app/mock/285595/post-students",  //请求的服务器url地址
            method: "post",
            // data是作为请求主体被发送的数据 data用于post、put、patch请求
            //默认是请求是application/json
            data: {p1: "A", p2: "B"} //请求的数据
        }).then((response) => {
            students.value = response.data.students
        }).catch((e) => {
            console.log(e)
        })
    }
    const sendAxiosPost2 = () => {
        axios({
            url: "http://rap2api.taobao.org/app/mock/285595/post-students",  //请求的服务器url地址
            method: "post",
            //修改Content-Type类型
            header: {"Content-Type": "application/x-www-form-urlencoded"},
            //qs.stringify方法把json对象序列化成url形式，以&拼接
            data: qs.stringify({p1: "A", p2: "B"}) //请求的数据
        }).then((response) => {
            students.value = response.data.students
        }).catch((e) => {
            console.log(e)
        })
    }
    const sendAxiosPost3 = () => {
        //Qs.stringify() 把json对象序列化url格式，用&连接  p1=A&p2=B
        const data = qs.stringify({
            p1: "A",
```

```
            p2: "B"
        });
        axios.post("http://rap2api.taobao.org/app/mock/285595/post-students",
data, {
                //修改Content-Type类型
                header: {"Content-Type": "application/x-www-form-urlencoded"}}
        ).then((response) => {
            students.value = response.data.students
        }).catch((e) => {
            console.log(e)
        })
    }
</script>

<style scoped>

</style>
```

5.ElementUI

网址：https://element-plus.org/zh-CN/#/zh-CN

vue项目中执行命令安装element-plus

main.js使用ElementPlus

```
import {createApp} from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'

const app = createApp(App);
//使用ElementPlus
app.use(ElementPlus)
app.use(store).use(router).mount('#app');
```

登录页面

```
<template>
    <el-form :model="ruleForm" :rules="rules" ref="ruleForm1" label-
width="100px">
        <el-form-item label="用户名" prop="username">
            <el-input v-model="ruleForm.username"></el-input>
        </el-form-item>
        <el-form-item label="密码" prop="password">
            <el-input v-model="ruleForm.password" type="password"></el-input>
        </el-form-item>
        <el-form-item label="记住密码" prop="remember">
            <el-switch v-model="ruleForm.remember"></el-switch>
        </el-form-item>
        <el-form-item>
```

```
            <el-button type="primary" @click="submitForm()">登录</el-button>
            <el-button @click="resetForm">重置</el-button>
        </el-form-item>
    </el-form>
</template>

<script setup>
    import {reactive,ref} from 'vue'

    const ruleForm1 = ref()
    const ruleForm = reactive({
        username: '',
        password: '',
        remember: false
    })
    const rules = reactive({
        username: [{
            required: true,
            message: '请输入用户名',
            trigger: 'blur'
        }, {
            min: 3,
            max: 5,
            message: '长度在 3 到 5 个字符',
            trigger: 'blur'
        }],
        password: [{
            required: true,
            message: '请输入密码',
            trigger: 'blur'
        }, {
            min: 5,
            max: 10,
            message: '长度在 5 到 10 个字符',
            trigger: 'blur'
        }]
    })
    const submitForm = () => {
        console.log(ruleForm1.value.validate)
        ruleForm1.value.validate((valid)=>{
            if(valid){
                //axios请求后端
                console.log(ruleForm)
                alert('验证通过')
            }else {
                alert('error submit!');
                return false;
            }
        })
    }
    const resetForm=()=>{
        ruleForm1.value.resetFields()
    }
</script>
```

```
<style scoped>

</style>
```