# 一、组件

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。在较高层面上，组件是自定义元素，Vue.js 的编译器为它添加特殊功能。组件其实就是一个拥有样式、动画、js逻辑、HTML结构的综合块。

- 如何划分组件

```
- 功能模块: select、pagenation...
- 页面区域划分: header、footer、sidebar...
```

- data必须是函数 通过 Vue 构造器传入的各种选项大多数都可以在组件里用。data 是一个例外，它必须是函数。
- 单文件组件的使用方式介绍 【项目开发都会是一个文件对应一个组件】 通过上面我们定义组件的方式，就已经感觉很不爽了，尤其是模板的定义，而且样式怎么处理也没有很好的进行规整。 Vue可以通过Webpack等第三方工具实现单文件的开发的方式。

```
// Hello.vue组件
<template>
    <h3>
        标题: {{str}}
    </h3>
</template>

<script>
    export default {
        data: function(){
            return {
                str: 'hello vue!'
            }
        }
    }
</script>

<style>
    h3{
        text-align: center;
    }
</style>
```

```
<script>
  // App.vue组件中使用Hello.vue组件
  // 导入Hello组件
  import Hello from './components/Hello';
  export default {
    // ...
    components: {  // 局部注册组件
      Hello
    }
  }
</script>
```

```
<template>
   // 使用自定义元素(Hello组件)
  <hello></hello>
</template>
```

> 通过vue-cli脚手架构造项目

- 组件间调用 --- components 如果要使用某个组件，单纯通过`import`导入是不行的，还需要通过`components`进行注册才能使用。

```
  // App.vue中

  // 导入组件
  import Header from './header'
  import Footer from './footer'

  new Vue({
    // 组件需要注册之后才能使用
    components:{
      Header, Footer
    }
  })

  // 在App.vue中
  <header></header>
  <footer></footer>
```

- 组件间通信 --- props (父组件给子组件传参) 组件实例的作用域是孤立的。这意味着不能 (也不应该) 在子组件的模板内直接引用父组件的数据。要让子组件使用父组件的数据，我们需要通过子组件的 props 选项。

```
    // header.vue 子组件
    new Vue({
      props: ['message'],  // 不同组件中的数据操作()
      methods: {
        doThis: function(){  // 子组件获取到父组件传递的参数this.message
          console.log(this.message);
        }
      }
    })
```

```
    // props可以其他写法
  props:{
      seller:{  // 即是设置接收参数的数据类型
        type: Object,  // 参数的类型
        // default 是设置默认值的
      }
   }
```

```
    // app.vue 父组件
    // 将字符串内容'hello world!'传递给子组件
    <header message='hello world!'></header>

    // 绑定data属性值
    // <header :message='title'></header>
```

- 组件间通信 --- 自定义事件(子组件给父组件传参)

```
  // app.vue 父组件

  // 自定义事件v-on: child-tell-me, 事件名为'child-tell-me'
  <component-b v-on:child-tell-me='getMsg'></component-b>

  new Vue({
    // ...
    methods: {
      // 'child-tell-me'对应触发的方法，即父组件获取子组件传递的参数msg
      getMsg: function(msg){
        console.log(msg)
      }
    }
})
```

```
  // footer.vue 子组件
    new Vue({
    // ...
    methods: {
      // 在sendMsg方法中，即触发'child-tell-me'方法，并传递参数
      sendMsg: function(msg){
        this.$emit('child-tell-me', '你收到我发送的消息了吗?【componentB】')
      }
    }
})
```

# 六、Vue路由

对于前端来说，其实浏览器配合超级连接就很好的实现了路由功能。但是对于单页面应用来说，浏览器和超级连接的跳转方式已经不能适用， 所以各大框架纷纷给出了单页面应用的解决路由跳转的方案。Vue框架的兼容性非常好，可以很好的跟其他第三方的路由框架进行结合。当然官方也给出了路由的方案：vue-router; 建议还是用官方的最好，使用量也是最大，相对来说Vue框架的升级路由组件升级也会及时跟上，所以为了以后的维护和升级方便还是使用Vue自家的东西最好。

- vue-router的安装使用 1.CDN连接方式

  ```
  <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
  ```

  2.npm 安装

  ```
  npm install vue-router
  ```

- vue-router 入门

  ```
  1.引入vue和vue-router(如果配合npm和webpack的话可以直接作为一个模块导入即可)
    import VueRouter from 'vue-router'
    Vue.use(VueRouter)

  2.定义路由跳转的组件(或导入组件)
    var Foo = { template: '<div>foo</div>' }
    var Bar = { template: '<div>bar</div>' }


  3.定义路由规则对象
    // 每个路由path应该映射一个组件
    var routes = [
        {path: '/foo', component: Foo},
        {path: '/Bar', component: Bar}
    ];

  4.创建路由器对象
    // 创建 router 实例，然后传 `routes` 配置
  ```

```
  var router = new VueRouter({
      routes,
      // 选中后的类名 (默认值是router-link-active)
      linkActiveClass: 'active'
  })

5.创建和挂载根实例
  const app = new Vue({
      router
  }).$mount('#app');

6. 模板中编写路由跳转链接
  <div id="app">
        <p>
            <!-- 使用 router-link 组件来导航. -->
            <!-- 通过传入 `to` 属性指定链接. -->
            <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
            <router-link to="/foo">Go to Foo</router-link>
            <router-link to="/bar">Go to Bar</router-link>
        </p>
        <!-- 路由出口 -->
        <!-- 路由匹配到的组件将渲染在这里 -->
        <router-view></router-view>
  </div>
```