vue实例的生命周期

- 什么是生命周期：从Vue实例创建、运行、到销毁期间，总是伴随着各种各样的事件，这些事件，统称为生命周期！
- 生命周期钩子：就是生命周期事件的别名而已；
- 生命周期钩子 = 生命周期函数 = 生命周期事件
- 主要的生命周期函数分类：
- 创建期间的生命周期函数：
  - beforeCreate：实例刚在内存中被创建出来，此时，还没有初始化好 data 和 methods 属性
  - created：实例已经在内存中创建OK，此时 data 和 methods 已经创建OK，此时还没有开始编译模板
  - beforeMount：此时已经完成了模板的编译，但是还没有挂载到页面中
  - mounted：此时，已经将编译好的模板，挂载到了页面指定的容器中显示
- 运行期间的生命周期函数：
- beforeUpdate：状态更新之前执行此函数， 此时 data 中的状态值是最新的，但是界面上显示的数据还是旧的，因为此时还没有开始重新渲染DOM节点
- updated：实例更新完毕之后调用此函数，此时 data 中的状态值 和 界面上显示的数据，都已经完成了更新，界面已经被重新渲染好了！
- 销毁期间的生命周期函数：
- beforeDestroy：实例销毁之前调用。在这一步，实例仍然完全可用。
- destroyed：Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

什么是路由

1. 对于普通的网站，所有的超链接都是URL地址，所有的URL地址都对应服务器上对应的资源；
2. 对于单页面应用程序来说，主要通过URL中的hash(#号)来实现不同页面之间的切换，同时，hash有一个特点：HTTP请求中不会包含hash相关的内容；所以，单页面程序中的页面跳转主要用hash实现；
3. 在单页面应用程序中，这种通过hash改变来切换页面的方式，称作前端路由（区别于后端路由）；

在 vue 中使用 vue-router

1. 导入 vue-router 组件类库：

```
<!-- 1. 导入 vue-router 组件类库 -->
  <script src="./lib/vue-router-2.7.0.js"></script>
```

2. 使用 router-link 组件来导航

```
<!-- 2. 使用 router-link 组件来导航 -->
<router-link to="/login">登录</router-link>
<router-link to="/register">注册</router-link>
```

3. 使用 router-view 组件来显示匹配到的组件

```
<!-- 3. 使用 router-view 组件来显示匹配到的组件 -->
<router-view></router-view>
```

4. 创建使用`Vue.extend`创建组件

```
    // 4.1 使用 Vue.extend 来创建登录组件
    var login = Vue.extend({
      template: '<h1>登录组件</h1>'
    });

    // 4.2 使用 Vue.extend 来创建注册组件
    var register = Vue.extend({
      template: '<h1>注册组件</h1>'
    });
```

5. 创建一个路由 router 实例，通过 routers 属性来定义路由匹配规则

```
// 5. 创建一个路由 router 实例，通过 routers 属性来定义路由匹配规则
    var router = new VueRouter({
      routes: [
        { path: '/login', component: login },
        { path: '/register', component: register }
      ]
    });
```

6. 使用 router 属性来使用路由规则

```
// 6. 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      router: router // 使用 router 属性来使用路由规则
    });
```

在路由规则中定义参数

1. 在规则中定义参数：

```
{ path: '/register/:id', component: register }
```

2. 通过`this.$route.params`来获取路由中的参数：

```
var register = Vue.extend({
      template: '<h1>注册组件 --- {{this.$route.params.id}}</h1>'
    });
```

使用`children`属性实现路由嵌套

```
  <div id="app">
    <router-link to="/account">Account</router-link>

    <router-view></router-view>
  </div>
```

```
  <script>
    // 父路由中的组件
    const account = Vue.extend({
      template: `<div>
        这是account组件
        <router-link to="/account/login">login</router-link> |
        <router-link to="/account/register">register</router-link>
        <router-view></router-view>
      </div>`
    });

    // 子路由中的 login 组件
    const login = Vue.extend({
      template: '<div>登录组件</div>'
    });

    // 子路由中的 register 组件
    const register = Vue.extend({
      template: '<div>注册组件</div>'
    });

    // 路由实例
    var router = new VueRouter({
      routes: [
        { path: '/', redirect: '/account/login' }, // 使用 redirect 实现路由重定向
        {
          path: '/account',
          component: account,
          children: [ // 通过 children 数组属性，来实现路由的嵌套
            { path: 'login', component: login }, // 注意，子路由的开头位置，不要加 /
路径符
            { path: 'register', component: register }
          ]
        }
      ]
    });

    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {},
      methods: {},
      components: {
        account
      },
      router: router
    });
  </script>
```

定义Vue组件

什么是组件： 组件的出现，就是为了拆分Vue实例的代码量的，能够让我们以不同的组件，来划分不同的功能模块，将来我们需要什么样的功能，就可以去调用对应的组件即可；
组件化和模块化的不同：

- 模块化： 是从代码逻辑的角度进行划分的；方便代码分层开发，保证每个功能模块的职能单一；

- 组件化： 是从UI界面的角度进行划分的；前端的组件化，方便UI组件的重用；

父组件向子组件传值

1. 组件实例定义方式，注意：一定要使用`props`属性来定义父组件传递过来的数据

```
<script>
    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {
        msg: '这是父组件中的消息'
      },
      components: {
        son: {
          template: '<h1>这是子组件 --- {{finfo}}</h1>',
          props: ['finfo']
        }
      }
    });
  </script>
```

2. 使用`v-bind`或简化指令，将数据传递到子组件中：

```
<div id="app">
    <son :finfo="msg"></son>
  </div>
```

子组件向父组件传值

1. 原理：父组件将方法的引用，传递到子组件内部，子组件在内部调用父组件传递过来的方法，同时把要发送给父组件的数据，在调用方法的时候当作参数传递进去；
2. 父组件将方法的引用传递给子组件，其中，`getMsg`是父组件中`methods`中定义的方法名称，`func`是子组件调用传递过来方法时候的方法名称

```
<son @func="getMsg"></son>
```

3. 子组件内部通过`this.$emit('方法名', 要传递的数据)`方式，来调用父组件中的方法，同时把数据传递给父组件使用

```
<div id="app">
    <!-- 引用父组件 -->
    <son @func="getMsg"></son>

    <!-- 组件模板定义 -->
    <script type="x-template" id="son">
      <div>
        <input type="button" value="向父组件传值" @click="sendMsg" />
      </div>
    </script>
  </div>

  <script>
```

```
    // 子组件的定义方式
    Vue.component('son', {
      template: '#son', // 组件模板Id
      methods: {
        sendMsg() { // 按钮的点击事件
          this.$emit('func', 'OK'); // 调用父组件传递过来的方法，同时把数据传递出去
        }
      }
    });

    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {},
      methods: {
        getMsg(val){ // 子组件中，通过 this.$emit() 实际调用的方法，在此进行定义
          alert(val);
        }
      }
    });
  </script>
```