Vue入门项目练习-ToDoList

后续我们通过一个小案例来继续了解Vue中的基础知识。

这个案例可以看成一个待办事项列表，包含的功能有：列出任务，设置当前任务，删除任务，增加任务。

准备工作，把HTML元素解构和CSS样式准备好

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>todo list</title>
  <style>
    #todoapp {
      width: 300px;
      height: 450px;
      border: 3px solid gold;
      margin: auto;
    }

    .main ul {
      list-style: none;
      width: 200px;
      margin: 20px auto;
    }

    .main ul>li {
      width: 150px;
      height: 20px;
      margin: 5px auto;
    }

    .view span {
      width: 20px;
      margin: auto;
    }

    h1 {
      text-align: center;
    }

    .header input {
      display: block;
      margin: auto;
    }

    .clear-completed {
      float: right;
    }
  </style>
```

```
</head>

<body>
  <div id='todoapp'>
    <div class="header">
      <h1>小海记事</h1>
      <input type="text" placeholder="请输入任务">
    </div>
    <div class="main">
      <ul class="todo-list">
        <li class="todo">任务1</li>
        <li class="todo">任务2</li>
        <li class="todo">任务3</li>
      </ul>
    </div>
    <div class="footer">
      <span class="todo-count">3</span>
      <button class="clear-completed">清空</button>
    </div>
  </div>
</body>
</html>
```

列出任务

首先开发最简单的功能--列出任务

脚本部分

```
    //创建vue对象
    const app = Vue.createApp({
      data(){
        todo_list = ['吃饭','睡觉','上班','学习']
        return {
          todo_list
        }
      }
    })
    const vm = app.mount('#todoapp')
```

页面部分

```
  <div id='todoapp'>
    <header>
      <h1>小海记事</h1>
      <input type="text" placeholder="请输入任务">
    </header>
    <div class="main">
      <ul class="todo-list">
        <li class="todo" v-for="item in todo_list">{{item}}</li>
      </ul>
    </div>
    <div class="footer">
      <span class="todo-count">3</span>
      <button class="clear-completed">清空</button>
```

```
    </div>
  </div>
```

增加任务

现在设计我们的需求，在任务输入框输入任务，回车之后就把任务添加到列表中，同时需要判断任务输入不为空，还要任务输入之后清空文本框

```
<body>
  <div id='todoapp'>
    <header>
      <h1>小海记事</h1>
      <input type="text" placeholder="请输入任务" v-model.trim="task"
@keyup.enter="add(task)">
    </header>
    <div class="main">
      <ul class="todo-list">
        <li class="todo" v-for="item in todo_list">{{item}}</li>
      </ul>
    </div>
    <div class="footer">
      <span class="todo-count">3</span>
      <button class="clear-completed">清空</button>
    </div>
  </div>

  <script src="../vue.js"></script>
  <script>

    //创建vue对象
    const app = Vue.createApp({
      data(){
        todo_list = ['吃饭','睡觉','上班','学习']
        task=''
        return {
          todo_list,
          task
        }
      },
      methods:{
        add(task){
          if(task){
            this.todo_list.push(task)  // 将任务保持到任务列表
            this.task=''               // 清空文本框
          }
        }
      }
    })
   const vm = app.mount('#todoapp') // 返回组件实例
  </script>
</body>
```

清空任务

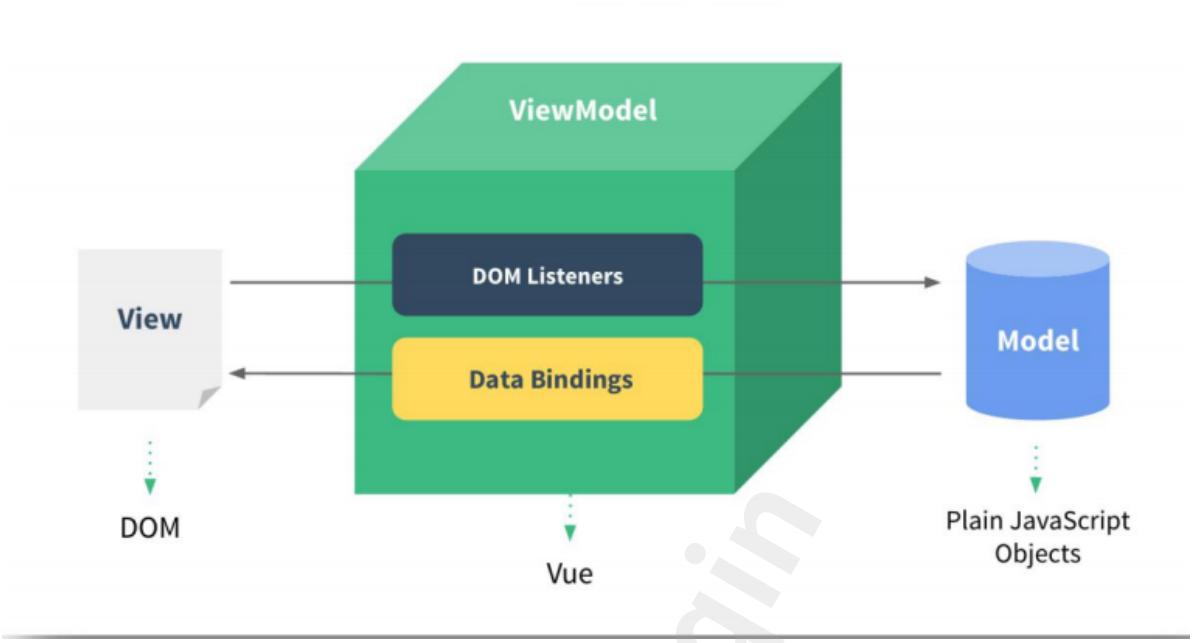
当我们点击清空按钮，希望清空当前任务列表

```
<body>
  <div id='todoapp'>
    <header>
      <h1>小海记事</h1>
      <input type="text" placeholder="请输入任务" v-model.trim="task"
@keyup.enter="add(task)">
    </header>
    <div class="main">
      <ul class="todo-list">
        <li class="todo" v-for="item in todo_list">{{item}}</li>
      </ul>
    </div>
    <div class="footer">
      <span class="todo-count">3</span>
      <button class="clear-completed" @click="empty">清空</button>
    </div>
  </div>

  <script src="../vue.js"></script>
  <script>

    //创建vue对象
    const app = Vue.createApp({
      data(){
        todo_list = ['吃饭','睡觉','上班','学习']
        task=''
        return {
          todo_list,
          task
        }
      },
      methods:{
        add(task){
          if(task){
            this.todo_list.push(task)  // 将任务保持到任务列表
            this.task=''               // 清空文本框
          }
        },
        empty(){
          this.todo_list=[]  //清空列表
        }
      }
    })
   const vm = app.mount('#todoapp') // 返回组件实例
  </script>
</body>
```


Vue技术扩展

MVVM模型

1. M：模型(Model)： 对应 data 中的数据
2. V：视图(View)： 模板
3. 视图模型(ViewModel)： Vue 实例对象



`const vm = app.mount('#todoapp')` 获得组件实例

vm可以拿到data中的数据

```
vm.todo_list  //Proxy {0: "吃饭", 1: "睡觉", 2: "上班", 3: "学习"}
vm.todo_list.push('新任务') // 往列表追加新任务，页面会同步
```

响应式原理

大家也看到了，我们获取的vm（组件实例）是一个代理，从vm获取到的todo_list也是一个代理。

那么代理是什么以及代理是如何工作的，到目前为止我们还不了解。

其实代理是ES6新增的扩展类，作用就是在操作目标对象之前，架设一层拦截，那么我们就可以知道用户对目标对象的操作，从而对其开发一些额外的功能。

Proxy(代理)

Proxy的本意为代理，即对目标对象的操作必须经过该代理，我们拿到这个代理就可以拦截用户对目标对象的操作行为，从而实现监控操作，Vue3响应式的底层原理就采用了Proxy，下面我们来看下具体使用方法。

```
const obj = {}
const p = new Proxy(obj,{
  get: function (){
    console.log('拦截get操作')
  },
  set: function () {
    console.log('拦截set操作')
  }
})
console.log(p)
p.a              //拦截get操作
p.a = 'haiwen'   //拦截set操作
```

上面代码对一个空对象架设了一层拦截，重定义了属性的读取（`get`）和设置（`set`）行为。我们在操作代理对象p时，实际上调用的是get和set对应的方法。代码还说明了Proxy 实际上重载（overload）了点运算符，即用自己的定义覆盖了语言的原始定义。

注意上述代码想要发生拦截效果，必须操作代理对象p，如果直接操作obj是没有拦截效果的。

```
obj.a='haiwen'
obj.a            //haiwen
```

上面的代码演示了拦截效果，但是把我们真正想要的功能给搞没了，如果我们既想保持拦截效果，又想保持原有功能需要：

```
const obj = {}
const p = new Proxy(obj,{
  get: function (target,key,){
    console.log('拦截get操作')
    return obj.key
  },
  set: function (target,key,value) {
    console.log('拦截set操作')
    obj.key=value
  }
})
console.log(p)
p.a='haiwen'  //拦截set操作  "haiwen"
p.a           //拦截get操作  "haiwen"
```

上面代码的get多了两个参数，target可以理解为目标对象，key可以理解为要操作的属性名

同理，set中的target，key也是如此，至于value当然就是所赋的值

Proxy的应用场景

如果我们把以上代码的打印语句替换成dom操作，如：

```
<h1 id="demo">hello</h1>
<script>
  const obj = {}
  const p = new Proxy(obj,{
    get: function (target,key,){
      console.log('拦截get操作')
      return obj.key
    },
```

```
    set: function (target,key,value) {
      console.log('拦截set操作')
      document.getElementById('demo').innerText=value
      obj.key=value
    }
  })
  console.log(p)

</script>
```

此时操作代理对象，不用刷新浏览器就发现页面元素发生变化

```
p.a='haiwen'
```

这个就是Vue3实现响应式操作的底层原理

Reflect（反射）

Reflect是配合Proxy而推出的另一个新的API，常常和Proxy一起使用，用于动态设置对象属性，比如前面我们设置对象属性采用的是点语法，实际这样做会出现一些意想不到的bug，如key传递的非字符串。因此使用Reflect操作帮助我们很好的处理这些问题。

```
  const obj = {}
  const p = new Proxy(obj,{
    get: function (target,key,){
      console.log('拦截get操作')
      return Reflect.get(target,key)
    },
    set: function (target,key,value) {
      console.log('拦截set操作')
      document.getElementById('demo').innerText=value
      Reflect.set(target,key,value)
    }
  })
  console.log(p)
```

选项式API扩展

目前我们学习通过选项来配置组件，如Data,methods。这种方式叫做选项式API，除此以外还有另外两个常用的选项。分别是computed和watch

计算属性

当处理复杂数据时，相比直接在插值语法里写表达式，更好的做法是写computed属性。

没用computed属性

```
<body>
  <div id="root">
    <h3>{{ate?'吃过了':'没吃呢'}}</h3>
  </div>
  <script src="../vue.js"></script>
  <script>
```

```
    const vm = Vue.createApp({
      data(){
        return {
          ate: false
        }
      }
    }).mount('#root')
  </script>
</body>
```

用了computed属性

```
<body>
  <div id="root">
    <h3>{{chilema}}</h3>
  </div>
  <script src="../vue.js"></script>
  <script>
    const vm = Vue.createApp({
      data(){
        return {
          ate: true
        }
      },
      computed:{
        chilema(){
          return this.ate?'吃过了':'没吃呢'
        }
      }
    }).mount('#root')
  </script>
</body>
```

模板的逻辑简洁了，复杂的逻辑写在代码中了，通过computed封装了。

计算属性与方法的区别

你可能发现，我把chilema放在methods里也能实现同样的效果，这好像并没有什么区别。

```
      methods:{
        chilema(){
          return this.ate?'吃过了':'没吃呢'
        }
      }
```

其实这里是有区别的，简单来说computed会缓存结果，methods会实时求职。

```
<body>
  <div id="root">
    <button @click="now">当前时间方法</button>
    <button @click="now2">当前时间计算</button>

  </div>
  <script src="../vue.js"></script>
  <script>
    const vm = Vue.createApp({
```

```
      data(){
        return {
          ate: true
        }
      },
      methods:{
        now(){
          console.log('方法',Date.now())
        }
      },
      computed:{
        now2(){
          console.log('计算',Date.now())
        }
      }
    }).mount('#root')
  </script>
</body>
```

可以看到，方法会每次调用，而计算属性只调用一次。

监听属性

监听器的作用：可以对数据的变化进行监听，并执行复杂逻辑。

语法：

```
监测的变量名(new变量名,old变量名){
  if (new变量名 xxx条件){
    //执行xxxx动作
    }
}
```

```
<body>
  <div id="root">
    <input type="text" placeholder="请输入用户名" v-model="username">
    <h3>{{result}}</h3>
  </div>
  <script src="../vue.js"></script>
  <script>
    const vm = Vue.createApp({
      data(){
        return {
          username: '',
          result:''
        }
      },
      watch:{
        //监听username的变化
        username(newName,oldName){
          console.log('监听变化')
          if(newName.length<8){
            this.result = '用户名不合法'
          }else{
            this.result = '合法'
```

```
          }
        }
      }
    }).mount('#root')
  </script>
</body>
```