Vue3训练营 - Element3.0开源实战

带你参加开源项目

- ## 1. Vue3.0 API 与 Element3.0组件开发实战
    - Vue3.0新语法基础
    - Compisiton-Api优势
    - Element组件的Vue3.0API更新实战
    - Vue3处理流程剖析 - 编译器、响应式数据、渲染函数
- ## 2. Vue3.0 原理剖析
    - 响应式Ractivity原理
    - 编译器原理
    - 渲染函数原理
    - Element3.0复杂特性移植
- ## 3. Element3.0项目的自动化测试、工程化、CI/CD
    - Jest测试
    - VTU
    - Element3.0单元测试实战
    - Github版本控制
    - Githook 提交信息校验
    - Eslint
    - 自动回归测试
    - GitHub Action完成CI/CD

第一天 Element组件实战

收获

- 能够使用Vue3.0 CompositionAPI
- 能够完成一个完整的PR过程
- 了解响应式原理

Vite环境搭建

https://github.com/vitejs/vite

```
yarn create vite-app vite
cd vite
yarn
yarn dev
```

Vue3官方CompositionAPI

https://v3.vuejs.org/guide/migration/render-function-api.html#overview

Vue3APi

https://v3.vuejs.org/

CompositionAPI详解

> 官方API https://composition-api.vuejs.org/api.htm

不断变化的数据

```
export default {
  name: "App",
  setup() {
    const position = reactive({});

    // 设置不断变化的数据
    window.addEventListener("mousemove", (e) => {
      console.log("mousemove:", e.pageX, e.pageY);

      position.x = e.pageX;
      position.y = e.pageY;
    });

    return { position };
  },
};
```

创建响应式数据

```
export default {
  name: "App",
  setup() {
    // 创建响应式数据
    const position = reactive({});

    // 设置不断变化的数据
    window.addEventListener("mousemove", (e) => {
      console.log("mousemove:", e.pageX, e.pageY);

      position.x = e.pageX;
      position.y = e.pageY;
    });

    return { position };
  },
};
```

定义渲染视图

```
<template>
  <div>Hello</div>
  <h1>x: {{position.x}} y: {{position.y}}</h1>
</template>
```

计算属性

```
const color = computed(() => {
  const hex = (num) => (num % 255).toString(16);
  return `#${hex(position.x) + hex(position.x) + '00'}`;
});

return { position ,color };
```

```
<h1 :style="{ background:color }">x : {{position.x}} y: {{position.y}}</h1>
```

ref拆装箱

- ref 将给定的值(确切的说是基本数据类型 ini 或 string)创建一个响应式的数据对象

```
const time = ref(0)
setInterval(() => {
  time.value = Date.now()
},1000)
```

数据监听

```
watch(
  time,
  (val, prev) => {
    console.log(`watch ${val}`);
  }
);
```

响应式副作用

```
watchEffect(() => {
  console.log("time", time.vlaue, unref(time));
});
```

事件

```
 const click = () => {
   time.value = 0;
 };
```

```
<button @click="click">Clear</button>
```

Element3.0源码实战

> 跟我一起编写Vue3版ElementUI
>
> https://juejin.im/post/6866373381424414734

获取历史版本过程

```
git log --oneline packages/button

##################################
commit 2d14bf977e986d49db7c8123a17638ae8f8004f7
Author: cuixiaorui <cui_xiaorui@126.com>
Date:   Thu Jul 30 17:43:21 2020 +0800

    refactor: use composition api refactoring button
##################################

##################################
commit 5136af71c1cc89d9c16fadf73f86ecdc76694d4a
Author: shengxinjing <316783812@qq.com>
Date:   Sat Jul 25 09:35:31 2020 +0800
##################################

# 回退为修改前
git reset —hard 2d14b
git reset --hard HEAD^
```

能力提高 刻意练习

Button旧版本

```
<template>
  <button
    class="el-button"
    @click="handleClick"
    :disabled="buttonDisabled || loading"
    :autofocus="autofocus"
    :type="nativeType"
    :class="[
      type ? 'el-button--' + type : '',
      buttonSize ? 'el-button--' + buttonSize : '',
```

```
      {
        'is-disabled': buttonDisabled,
        'is-loading': loading,
        'is-plain': plain,
        'is-round': round,
        'is-circle': circle
      }
    ]"
  >
    <i class="el-icon-loading" v-if="loading"></i>
    <i :class="icon" v-if="icon && !loading"></i>
    <span v-if="$slots.default"><slot></slot></span>
  </button>
</template>
<script>
  export default {
    name: 'ElButton',

    inject: {
      elForm: {
        default: ''
      },
      elFormItem: {
        default: ''
      }
    },

    props: {
      type: {
        type: String,
        default: 'default'
      },
      size: String,
      icon: {
        type: String,
        default: ''
      },
      nativeType: {
        type: String,
        default: 'button'
      },
      loading: Boolean,
      disabled: Boolean,
      plain: Boolean,
      autofocus: Boolean,
      round: Boolean,
      circle: Boolean
    },
```

```
    computed: {
      _elFormItemSize() {
        return (this.elFormItem || {}).elFormItemSize;
      },
      buttonSize() {
        return this.size || this._elFormItemSize || (this.$ELEMENT || {}).size;
      },
      buttonDisabled() {
        return this.disabled || (this.elForm || {}).disabled;
      }
    },

    methods: {
      handleClick(evt) {
        this.$emit('click', evt);
      }
    }
  };
</script>
```

过程版本

```
import { computed, inject, toRefs, unref, getCurrentInstance } from 'vue'

size: {
  type: String,
    default: ''
},


setup(props, ctx) {
    const { size, disabled } = toRefs(props);
    const elFormItem = inject("elFormItem", {});
    const elForm = inject("elForm", {});
    const _elFormItemSize = computed(() => unref(elFormItem.elFormItemSize));

    const buttonSize = computed(
      () =>
        size.value ||
        _elFormItemSize.value ||
        (getCurrentInstance().proxy.$ELEMENT || {}).size
    );
    const buttonDisabled = computed(
      () => disabled.value || unref(elForm.disabled)
    );
```

```
    const handleClick = (evt) => {
      ctx.emit("click", evt);
    };

    return {
      buttonSize,
      buttonDisabled,
      handleClick,
    };
},
emits: ['click'],
```

Button完全版本

```
<template>
  <button
    class="el-button"
    @click="handleClick"
    :disabled="buttonDisabled || loading"
    :autofocus="autofocus"
    :type="nativeType"
    :class="[
      type ? 'el-button--' + type : '',
      buttonSize ? 'el-button--' + buttonSize : '',
      {
        'is-disabled': buttonDisabled,
        'is-loading': loading,
        'is-plain': plain,
        'is-round': round,
        'is-circle': circle
      }
    ]"
  >
    <i class="el-icon-loading" v-if="loading"></i>
    <i :class="icon" v-if="icon && !loading"></i>
    <span v-if="$slots.default">
      <slot></slot>
    </span>
  </button>
</template>
<script>
import { computed, inject, toRefs, unref, getCurrentInstance } from 'vue'

export default {
  name: 'ElButton',
```

```
  props: {
    type: {
      type: String,
      default: 'default'
    },
    size: {
      type: String,
      default: ''
    },
    icon: {
      type: String,
      default: ''
    },
    nativeType: {
      type: String,
      default: 'button'
    },
    loading: Boolean,
    disabled: Boolean,
    plain: Boolean,
    autofocus: Boolean,
    round: Boolean,
    circle: Boolean
  },
  emits: ['click'],
  setup(props, ctx) {
    const { size, disabled } = toRefs(props)

    const buttonSize = useButtonSize(size)
    const buttonDisabled = useButtonDisabled(disabled)

    const handleClick = (evt) => {
      ctx.emit('click', evt)
    }

    return {
      handleClick,
      buttonSize,
      buttonDisabled
    }
  }
}

const useButtonSize = (size) => {
  const elFormItem = inject('elFormItem', {})

  const _elFormItemSize = computed(() => {
    return unref(elFormItem.elFormItemSize)
  })
```

```
  const buttonSize = computed(() => {
    return (
      size.value ||
      _elFormItemSize.value ||
      (getCurrentInstance().proxy.$ELEMENT || {}).size
    )
  })

  return buttonSize
}

const useButtonDisabled = (disabled) => {
  const elForm = inject('elForm', {})

  const buttonDisabled = computed(() => {
    return disabled.value || unref(elForm.disabled)
  })

  return buttonDisabled
}
</script>
```

单元测试

Button.spec.js

```
import Button from '../Button.vue'
import { mount } from '@vue/test-utils'
describe('Button.vue', () => {
  describe('props', () => {
    it('type', () => {
      const wrapper = mount(Button, {
        props: {
          type: 'primary'
        }
      })

      expect(wrapper.classes()).toContain('el-button--primary')
    })

    it('icon', () => {
```

```
    const wrapper = mount(Button, {
      props: {
        icon: 'el-icon-search'
      }
    })

    expect(wrapper.classes()).toContain('el-button--default')
  })

  it('nativeType', () => {
    const wrapper = mount(Button, {
      props: {
        nativeType: 'submit'
      }
    })

    expect(wrapper.attributes('type')).toBe('submit')
  })

  it('loading', () => {
    const wrapper = mount(Button, {
      props: {
        loading: true
      }
    })

    expect(wrapper.classes()).toContain('is-loading')
    expect(wrapper.find('.el-icon-loading').exists()).toBe(true)
  })

  it('disabled', () => {
    const wrapper = mount(Button, {
      props: {
        disabled: true
      }
    })

    expect(wrapper.classes()).toContain('is-disabled')
  })

  it('size', () => {
    const wrapper = mount(Button, {
      props: {
        size: 'medium'
      }
    })

    expect(wrapper.classes()).toContain('el-button--medium')
  })
```

```
  it('plain', () => {
    const wrapper = mount(Button, {
      props: {
        plain: true
      }
    })

    expect(wrapper.classes()).toContain('is-plain')
  })

  it('round', () => {
    const wrapper = mount(Button, {
      props: {
        round: true
      }
    })

    expect(wrapper.classes()).toContain('is-round')
  })

  it('circle', () => {
    const wrapper = mount(Button, {
      props: {
        circle: true
      }
    })

    expect(wrapper.classes()).toContain('is-circle')
  })
})

it('captures click events emitted via click', () => {
  const wrapper = mount(Button)
  wrapper.trigger('click')

  expect(wrapper.emitted('click')).toBeTruthy()
  expect(wrapper.emitted('click').length).toBe(1)
})

it('should only will trigger a click event', async () => {
  let count = 0
  const Comp = {
    template: '<div><el-button @click="handleClick"></el-button></div>',
    setup() {
      const handleClick = () => count++
      return { handleClick }
    }
  }
```

```
    const wrapper = mount(Comp, {
      global: {
        components: {
          'el-button': Button
        }
      }
    })

    await wrapper.findComponent({ name: 'ElButton' }).trigger('click')

    expect(count).toBe(1)
  })

  it("can't captures click events emitted via click when loading ", () => {
    const wrapper = mount(Button, {
      props: {
        loading: true
      }
    })
    wrapper.trigger('click')

    expect(wrapper.emitted('click')).toBeFalsy()
  })
})
```

响应式原理

[渐进式手敲Vue3.0框架]

[Vue3新特性一篇搞懂]

响应式是概念

首先我们说说什么是响应式。通过某种方法可以达到数据变了可以自由定义对应的响应就叫响应式。

```
let effective
function effect(fun) {
    effective = fun
}

function reactive(data) {
    if (typeof data !== 'object' || data === null) {
        return data
    }
    const observed = new Proxy(data, {
        get(target, key, receiver) {
            // 普通写法
```

```
            // return target[key]
            // proxy + reflect 反射
            // Reflect有返回值不报错
            let result = Reflect.get(target, key, receiver)

            // return result
            // 多层代理
            return typeof result !== 'object' ? result : reactive(result)
        },
        set(target, key, value, receiver) {
            effective()
            // 普通写法
            // target[key] = value // 如果设置不成功 没有返回
            // proxy + reflect
            const ret = Reflect.set(target, key, value, receiver)
            return ret
        },

        deleteProperty(target,key){
            const ret = Reflect.deleteProperty(target,key)
            return ret
        }

    })
    return observed
}

module.exports = {
    reactive, effect
}
```

```
<template>
  <h1 :style="{ background:color }">x : {{position.x}} y: {{position.y}}</h1>
  <h1>{{new Date(time)}}</h1>
  <h1>{{new Date(position.time)}}</h1>
  <button @click="click">Clear</button>
</template>

<script>
import { reactive, computed, ref, unref, watchEffect, watch } from "vue";
export default {
  name: "App",

  setup() {
    const position = reactive({ x: 0, y: 0 });

    // 设置不断变化的数据
```

```
    window.addEventListener("mousemove", (e) => {
      // console.log("mousemove:", e.pageX, e.pageY);

      position.x = e.pageX;
      position.y = e.pageY;
    });

    const color = computed(() => {
      const hex = (num) => (num % 255).toString(16);
      return `#${hex(position.x) + hex(position.x) + "00"}`;
    });

    const { time, click } = useTime();

    function useTime() {
      // ref
      const time = ref(0);
      setInterval(() => {
        time.value = Date.now();
      }, 1000);

      position.time = time;

      // watchEffect
      watchEffect(() => {
        console.log("time", time.vlaue, unref(time));
      });

      // watch
      watch(
        time,
        (val, prev) => {
          console.log(`watch ${val}`);
        } // getter
      );

      const click = () => {
        time.value = 0;
      };
      return { time, click };
    }

    return { position, color, time, click };
  },
};

module.exports = {
  useTime: useTime,
  useTime: useTime,
```

```
};
</script>
```