/ Render 函数

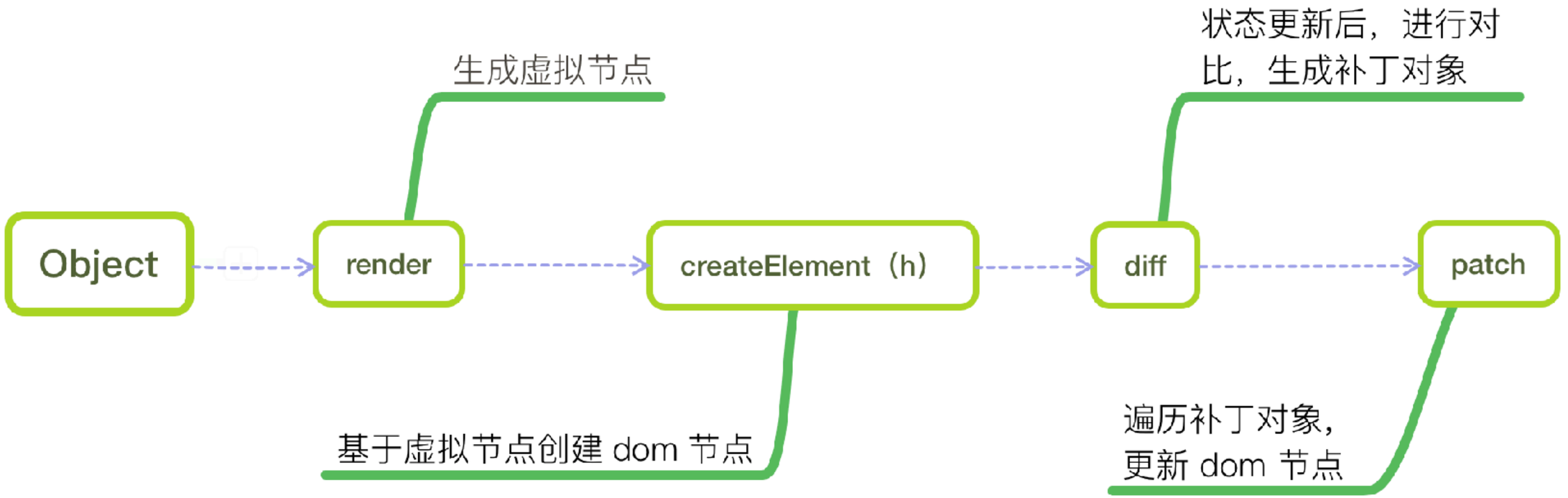
或加微信：talkingcoder，
注明 Render

关于我

Aresn

- TalkingData
- TalkingCoder
- iView
- 《Vue.js 实战》

什么是 Virtual Dom



```html
<div id="main">
    <p>文本内容</p>
    <p>文本内容</p>
</div>
```

普通 DOM

```js
var vNode = {
    tag: 'div',
    attributes: {
        id: 'main'
    },
    children: [
        // p 节点
    ]
}
```

Virtual DOM

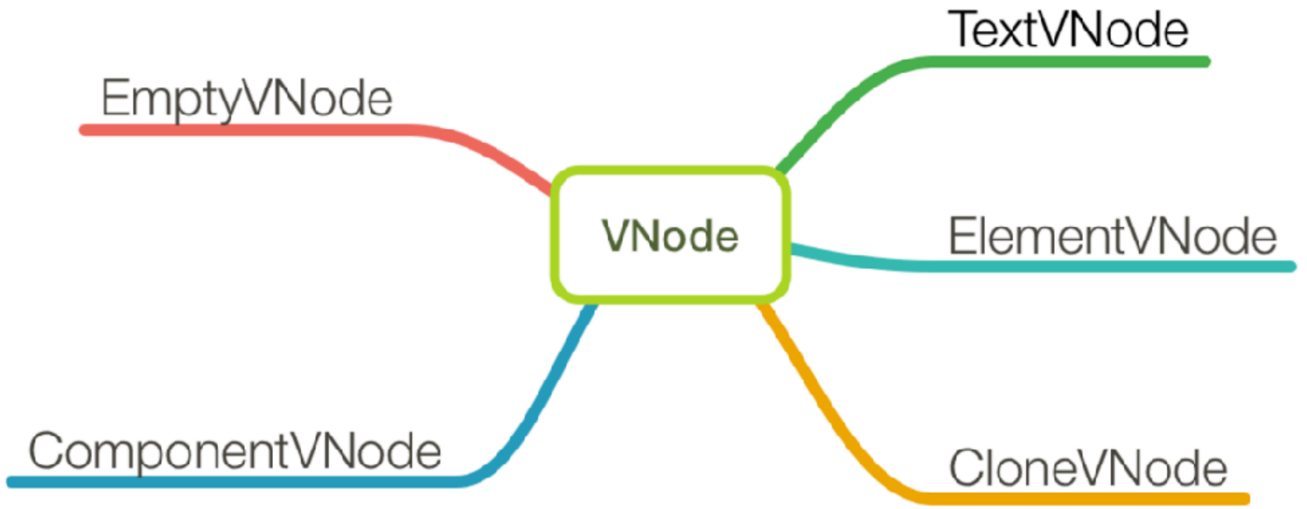
Vue2 的 VNode

tag：当前节点的标签名
data：当前节点的数据对象（下一页）
children：子节点，数组，也是 VNode 类型
text：当前节点的文本，一般文本节点或注释节点会有该属性
elm：当前虚拟节点对应的真实的 DOM 节点
ns：节点的 namespace
content：编译作用域
functionalContext：函数化组件的作用域
key：节点的key属性，用于作为节点的标识，有利于 patch 的优化
componentOptions：创建组件实例时会用到的选项信息
child：当前节点对应的组件实例
parent：组件的占位节点
raw：原始 html
isStatic：静态节点的标识
isRootInsert：是否作为根节点插入，被 <transition> 包裹的节点，该属性的值为false
isComment：当前节点是否是注释节点
isCloned：当前节点是否为克隆节点
isOnce：当前节点是否有 v-once 指令

```ts
export interface VNode {
    tag?: string;
    data?: VNodeData;
    children?: VNode[];
    text?: string;
    elm?: Node;
    ns?: string;
    context?: Vue;
    key?: string | number;
    componentOptions?: VNodeComponentOptions;
    componentInstance?: Vue;
    parent?: VNode;
    raw?: boolean;
    isStatic?: boolean;
    isRootInsert: boolean;
    isComment: boolean;
}
```

源码中 VNode 的定义

VNode 主要类型



TextVNode：文本节点
ElementVNode：普通元素节点
ComponentVNode：组件节点
EmptyVNode：没有内容的注释节点
CloneVNode：克隆节点，可以是以上任意类型的节点，唯一的区别在于 isCloned 属性为 true

为什么用 Render 函数

普通组件在该场景下的问题

- 代码冗长
- template 中大部分代码是相同的
- 外层必须包含一个无用的

使用 Render 函数重构

createElement 用法（h）

第一个参数必选，可以是一个 HTML 标签，也可以是一个组件或函数；

第二个是可选参数，数据对象（具体见下页）

第三个是子节点，也是可选，用法一致

```
createElement(
    // {String | Object | Function}
    // 一个 HTML 标签，组件选项，或一个函数
    // 必须 Return 上述其中一个
    'div',
    // {Object}
    // 一个对应属性的数据对象，可选
    // 您可以在 template 中使用
    {
        // 稍后详细介绍
    },
    // {String | Array}
    // 子节点(VNodes)，可选
    [
        createElement('h1', 'hello world'),
        createElement(MyComponent, {
            props: {
                someProp: 'foo'
            }
        }),
        'bar'
    ]
)
```

Vue2 的 VNodeData

class：v-bind:class

style：v-bind:style

attrs：一般的 HTML 属性，比如 id

props：props

on：自定义事件

nativeOn：原生事件，比如 click

```ts
export interface VNodeData {
    key?: string | number;
    slot?: string;
    scopedSlots?: { [key: string]: ScopedSlot };
    ref?: string;
    tag?: string;
    staticClass?: string;
    class?: any;
    staticStyle?: { [key: string]: any };
    style?: Object[] | Object;
    props?: { [key: string]: any };
    attrs?: { [key: string]: any };
    domProps?: { [key: string]: any };
    hook?: { [key: string]: Function };
    on?: { [key: string]: Function | Function[] };
    nativeOn?: { [key: string]: Function | Function[] };
    transition?: Object;
    show?: boolean;
    inlineTemplate?: {
      render: Function;
      staticRenderFns: Function[];
    };
    directives?: VNodeDirective[];
    keepAlive?: boolean;
}
```

两种组件写法对比

```
Vue.component('ele', {
    template: '\
        <div id="element" \
        :class="{show: show}" \
        @click="handleClick">文本内容</div>',
    data: function () {
        return {
            show: true
        }
    },
    methods: {
        handleClick: function () {
            console.log('clicked!');
        }
    }
});
```

普通组件

```
Vue.component('ele', {
    render: function (createElement) {
        return createElement(
            'div',
            {
                class: {
                    'show': this.show
                },
                attrs: {
                    id: 'element'
                },
                on: {
                    click: this.handleClick
                }
            },
            '文本内容'
        )
    },
    data: function () {
        return {
            show: true
        }
    },
    methods: {
        handleClick: function () {
            console.log('clicked!');
        }
    }
});
```

Render 函数

约束

所有的组件树中，如果 VNode 是组件，或含有组件的 slot，那 VNode 必须唯一。

约束

错误的用法，因为子组件重复了

```
// 局部声明组件
    var Child = {
        render: function(createElement) {
            return createElement('p', 'text');
        }
    };
    Vue.component('ele', {
        render: function (createElement) {
            // 创建一个子节点，使用组件 Child
            var ChildNode = createElement(Child);
            return createElement('div', [
                ChildNode,
                ChildNode
            ]);
        }
    });
```

约束

错误的用法，因为 slot 包含了组件且重复了

```
<div id="app">
    <ele>
        <div>
            <Child></Child>
        </div>
    </ele>
</div>
<script>
    // 全局注册组件
    Vue.component('Child', {
        render: function (createElement) {
            return createElement('p', 'text');
        }
    });
    Vue.component('ele', {
        render: function (createElement) {
            return createElement('div', [
                this.$slots.default,
                this.$slots.default
            ]);
        }
    });

    var app = new Vue({
        el: '#app'
    })
</script>
```

怎么办?

组件的复用

```
// 局部声明组件
var Child = {
    render: function(createElement) {
        return createElement('p', 'text');
    }
};
Vue.component('ele', {
    render: function (createElement) {
        return createElement('div',
            Array.apply(null, {
                length: 5
            }).map(function() {
                return createElement(Child);
            })
        );
    }
});
```

怎么办?

slot的克隆

```
Vue.component('ele', {
    render: function (createElement) {
        // 克隆 slot 节点的方法
        function cloneVNode (vnode) {
            // 递归遍历所有子节点，并克隆
            const clonedChildren = vnode.children &&
            vnode.children.map(function(vnode) {
                return cloneVNode(vnode);
            });
            const cloned = createElement(
                vnode.tag,
                vnode.data,
                clonedChildren
            );
            cloned.text = vnode.text;
            cloned.isComment = vnode.isComment;
            cloned.componentOptions = vnode.componentOptions;
            cloned.elm = vnode.elm;
            cloned.context = vnode.context;
            cloned.ns = vnode.ns;
            cloned.isStatic = vnode.isStatic;
            cloned.key = vnode.key;

            return cloned;
        }

        const vNodes = this.$slots.default;
        const clonedVNodes = vNodes.map(function(vnode) {
            return cloneVNode(vnode);
        });

        return createElement('div', [
            vNodes,
            clonedVNodes
        ]);
    }
});
```

使用 JavaScript 代替模板功能

在 Render 函数中，没有指令，
一切都可以用 JS 来实现。

v-if

```
<div id="app">
    <ele :show="show"></ele>
    <button @click="show = !show">切换 show</button>
</div>
<script>
    Vue.component('ele', {
        render: function (createElement) {
            if (this.show) {
                return createElement('p', 'show 的值为 true');
            } else {
                return createElement('p', 'show 的值为 false');
            }
        },
        props: {
            show: {
                type: Boolean,
                default: false
            }
        }
    });

    var app = new Vue({
        el: '#app',
        data: {
            show: false
        }
    })
</script>
```

使用 JavaScript 代替模板功能

```
<div id="app">
    <ele :list="list"></ele>
</div>
<script>
    Vue.component('ele', {
        render: function (createElement) {
            var nodes = [];
            for (var i = 0; i < this.list.length; i++) {
                nodes.push(createElement('p', this.list[i]));
            }
            return createElement('div', nodes);
        },
        props: {
            list: {
                type: Array
            }
        }
    });

    var app = new Vue({
        el: '#app',
        data: {
            list: [
                '《Vue.js实战》',
                '《JavaScript高级程序设计》',
                '《JavaScript语言精粹》'
            ]
        }
    })
</script>
```

v-for

使用 JavaScript 代替模板功能

```
Vue.component('ele', {
    render: function (createElement) {
        if (this.list.length) {
            return createElement('ul', this.list.map(function (item) {
                return createElement('li', item);
            }));
        } else {
            return createElement('p', '列表为空');
        }
    },
    props: {
        list: {
            type: Array,
            default: function () {
                return [];
            }
        }
    }
});
```

综合

使用 JavaScript 代替模板功能

```
Vue.component('ele', {
    render: function (createElement) {
        var _this = this;
        return createElement('div', [
            createElement('input', {
                domProps: {
                    value: this.value
                },
                on: {
                    input: function (event) {
                        _this.value = event.target.value;
                    }
                }
            }),
            createElement('p', 'value: ' + this.value)
        ])
    },
    data: function () {
        return {
            value: ''
        }
    }
});
```

v-model

事件修饰符

修饰符	对应的句柄
.stop	event.stopPropagation()
.prevent	event.preventDefault()
.self	if (event.target !== event.currentTarget) return
.enter、.13	if (event.keyCode !== 13) return 替换13位需要的 keyCode
.ctrl、.alt、 .shift、 .meta	if (!event.ctrlKey) return 根据需要替换 ctrlKey 为 altKey、shiftKey 或 metaKey

事件修饰符前缀

修饰符	前缀
.capture	!
.once	~
.capture.once 或 .once.capture	~!

```
on: {
    '!click': this.doThisInCapturingMode,
    '~keyup': this.doThisOnce,
    `~!mouseover`: this.doThisOnceInCapturingMode
}
```

slot 的默认内容

this.$slots.default 等于 undefined，就说明父组件中没有定义 slot，这时可以自定义显示的内容。

```
<div id="app">
    <ele></ele>
    <ele>
        <p>slot 的内容</p>
    </ele>
</div>
<script>
    Vue.component('ele', {
        render: function (createElement) {
            if (this.$slots.default === undefined) {
                return createElement('div', '没有使用 slot 时显示的文本');
            } else {
                return createElement('div', this.$slots.default);
            }
        }
    });

    var app = new Vue({
        el: '#app'
    })
</script>
```

函数化组件

Vue.js 提供了一个 **functional** 的布尔值选项，设置为 true 可以使组件无状态和无实例，也就是没有 data 和 this 上下文。这样用 render 函数返回虚拟节点可以更容易渲染，因为函数化组件只是一个函数，渲染开销要小很多。

使用函数化组件时，Render 函数提供了第二个参数 context 来提供临时上下文。组件需要的 data、props、slots、children、parent 都是通过这个上下文来传递，比如 this.level 要改写为 context.props.level，this.$slots.default 改写为 context.children。

函数化组件

Demo

函数化组件

适用场景

- 程序化地在多个组件中选择一个
- 在将 children, props, data 传递给子组件之前操作它们

实战：使用 Render 函数开发可排序的表格组件

Demo

作业

- 练习1：查阅资料，了解表格的 <colgroup> 和 <col> 元素用法后，给 v-table 的 columns 增加一个可以设置列宽的 width 字段，并实现该功能。

- 练习2：将该示例的 render 写法改写为 template 写法，加以对比，总结出两者的差异性，深刻理解其使用场景。

总结

实战的练习题中，有用 template 写法还原 render 函数，目的是充分理解 render 函数的使用场景。在做了还原后，会发现使用 template 写法更简单、可读。的确，实战示例更适合用 template 来实现，在业务中，生产效率是第一位，所以绝大部分业务代码都应当用 template 来完成。你不用在意性能问题，如果使用了 webpack 做编译（后面章节会介绍），template 都会被预编译为 render 函数。

Q & A