链滴

从 Vue 源码分析 -Virtual DOM

作者：gmw-zjw
原文链接：https://ld246.com/article/1571059429067
来源网站：链滴

Virtual DOM

这是一个前端人人都接触的问题，都知道DOM操作是非常昂贵的，来看看vue的 Virtual DOM是怎实现的。

创建一个vnode：

```
// instance/vdom/vnode.js
/*
 * @Description: vnode 虚拟dom定义实现
 * @Author: 执念
 * @Date: 2019-08-14 10:42:22
 * @LastEditTime: 2019-10-14 21:07:48
 * @LastEditors: Please set LastEditors
 */
/* @flow */

export default class VNode {
  tag: string | void;
  data: VNodeData | void;
  children: ?Array<VNode>;
  text: string | void;
  elm: Node | void;
  ns: string | void;
  context: Component | void; // rendered in this component's scope
  key: string | number | void;
  componentOptions: VNodeComponentOptions | void;
  componentInstance: Component | void; // component instance
  parent: VNode | void; // component placeholder node

  // strictly internal
  raw: boolean; // contains raw HTML? (server only)
  isStatic: boolean; // hoisted static node
  isRootInsert: boolean; // necessary for enter transition check
  isComment: boolean; // empty comment placeholder?
  isCloned: boolean; // is a cloned node?
  isOnce: boolean; // is a v-once node?
  asyncFactory: Function | void; // async component factory function
  asyncMeta: Object | void;
  isAsyncPlaceholder: boolean;
  ssrContext: Object | void;
  fnContext: Component | void; // real context vm for functional nodes
  fnOptions: ?ComponentOptions; // for SSR caching
  devtoolsMeta: ?Object; // used to store functional render context for devtools
  fnScopeId: ?string; // functional scope id support

  constructor (
    tag?: string,
    data?: VNodeData,
    children?: ?Array<VNode>,
    text?: string,
    elm?: Node,
```

```
    context?: Component,
    componentOptions?: VNodeComponentOptions,
    asyncFactory?: Function
  ) {
    /* 当前节点的标签名 */
    this.tag = tag
    /* 当前节点对应的对象，包含了一些数据信息，是一个VNodeData类型 */
    this.data = data
    /* 当前节点的子节点, 是一个数组 */
    this.children = children
    /* 当前节点的文本 */
    this.text = text
    /* 当前虚拟节点对应的真实dom节点 */
    this.elm = elm
    /* 当前节点的命名空间 */
    this.ns = undefined
    /* 编译作用域 */
    this.context = context
    /* 函数化组件的作用域 */
    this.fnContext = undefined
    this.fnOptions = undefined
    this.fnScopeId = undefined
    /* 节点的key属性，被当做节点的标志，用作优化 */
    this.key = data && data.key
    /* 组件options选项 */
    this.componentOptions = componentOptions
    /* 当前节点对应的组件实例 */
    this.componentInstance = undefined
    /* 当前父节点 */
    this.parent = undefined
    /* 简而言之就是是否为原生HTML或只是普通文本，innerHTML的时候为true，textContent的时
为false */
    this.raw = false
    /* 静态节点标志 */
    this.isStatic = false
    /* 是否作为根节点插入 */
    this.isRootInsert = true
    /* 是否为注释节点 */
    this.isComment = false
    /* 是否为克隆节点 */
    this.isCloned = false
    /* 是否有v-once命令 */
    this.isOnce = false
    this.asyncFactory = asyncFactory
    this.asyncMeta = undefined
    this.isAsyncPlaceholder = false
  }

  // DEPRECATED: alias for componentInstance for backwards compat.
  /* istanbul ignore next */
  get child (): Component | void {
    return this.componentInstance
  }
}
```

```
/**
 * 创建一个空的vnode
 */
export const createEmptyVNode = (text: string = '') => {
  const node = new VNode()
  node.text = text
  node.isComment = true
  return node
}

/**
 * 创建一个文本节点
 */
export function createTextVNode (val: string | number) {
  return new VNode(undefined, undefined, undefined, String(val))
}

// optimized shallow clone
// used for static nodes and slot nodes because they may be reused across
// multiple renders, cloning them avoids errors when DOM manipulations rely
// on their elm reference.
// 复制一个 vnode
export function cloneVNode (vnode: VNode): VNode {
  const cloned = new VNode(
    vnode.tag,
    vnode.data,
    // #7975
    // clone children array to avoid mutating original in case of cloning
    // a child.
    vnode.children && vnode.children.slice(),
    vnode.text,
    vnode.elm,
    vnode.context,
    vnode.componentOptions,
    vnode.asyncFactory
  )
  cloned.ns = vnode.ns
  cloned.isStatic = vnode.isStatic
  cloned.key = vnode.key
  cloned.isComment = vnode.isComment
  cloned.fnContext = vnode.fnContext
  cloned.fnOptions = vnode.fnOptions
  cloned.fnScopeId = vnode.fnScopeId
  cloned.asyncMeta = vnode.asyncMeta
  cloned.isCloned = true
  return cloned
}
```

创建vnode实例，在构造函数中添加属性，

创建一个空vnode节点，并导出：

```
/**
 * 创建一个空的vnode
 */
export const createEmptyVNode = (text: string = '') => {
  const node = new VNode()
  node.text = text
  node.isComment = true
  return node
}
```

创建一个文本节点:

```
/**
 * 创建一个文本节点:
 */
export function createTextVNode (val: string | number) {
  return new VNode(undefined, undefined, undefined, String(val))
}
```

复制一个vnode:

```
// 复制一个 vnode
export function cloneVNode (vnode: VNode): VNode {
  const cloned = new VNode(
    vnode.tag,
    vnode.data,
    // #7975
    // clone children array to avoid mutating original in case of cloning
    // a child.
    vnode.children && vnode.children.slice(),
    vnode.text,
    vnode.elm,
    vnode.context,
    vnode.componentOptions,
    vnode.asyncFactory
  )
  cloned.ns = vnode.ns
  cloned.isStatic = vnode.isStatic
  cloned.key = vnode.key
  cloned.isComment = vnode.isComment
  cloned.fnContext = vnode.fnContext
  cloned.fnOptions = vnode.fnOptions
  cloned.fnScopeId = vnode.fnScopeId
  cloned.asyncMeta = vnode.asyncMeta
  cloned.isCloned = true
  return cloned
}
```

创建VDOM

创建vdom，返回一个vnode实例

```
export function createElement (
  context: Component,
```

```
  tag: any,
  data: any,
  children: any,
  normalizationType: any,
  alwaysNormalize: boolean
): VNode | Array<VNode> {
// 如果有节点，那么直接返回，否则创建新节点
  if (Array.isArray(data) || isPrimitive(data)) {
    normalizationType = children
    children = data
    data = undefined
  }
  if (isTrue(alwaysNormalize)) {
    normalizationType = ALWAYS_NORMALIZE
  }
  // 创建一个新的节点
  return _createElement(context, tag, data, children, normalizationType)
}
```

创建一个新的节点

```
export function _createElement (
  context: Component,
  tag?: string | Class<Component> | Function | Object,
  data?: VNodeData,
  children?: any,
  normalizationType?: number
): VNode | Array<VNode> {
  // 如果传递data参数且data的__ob__已经定义，则说明已经存在observer对象了
  // 创建一个空节点
  if (isDef(data) && isDef((data: any).__ob__)) {
    process.env.NODE_ENV !== 'production' && warn(
      `Avoid using observed data object as vnode data: ${JSON.stringify(data)}\n` +
      'Always create fresh vnode data objects in each render!',
      context
    )
    // 直接返回空节点
    return createEmptyVNode()
  }
  // object syntax in v-bind
  if (isDef(data) && isDef(data.is)) {
    tag = data.is
  }
  // 如果tag不存在则创建一个空节点
  if (!tag) {
    // in case of component :is set to falsy value
    return createEmptyVNode()
  }
  // warn against non-primitive key
  if (process.env.NODE_ENV !== 'production' &&
    isDef(data) && isDef(data.key) && !isPrimitive(data.key)
  ) {
    if (!__WEEX__ || !('@binding' in data.key)) {
      warn(
```

```
        'Avoid using non-primitive value as key, ' +
        'use string/number value instead.',
        context
      )
    }
  }
  // support single function children as default scoped slot
  // 默认默认的作用域插槽(slot)
  if (Array.isArray(children) &&
    typeof children[0] === 'function'
  ) {
    data = data || {}
    data.scopedSlots = { default: children[0] }
    children.length = 0
  }
  if (normalizationType === ALWAYS_NORMALIZE) {
    children = normalizeChildren(children)
  } else if (normalizationType === SIMPLE_NORMALIZE) {
    children = simpleNormalizeChildren(children)
  }
  let vnode, ns
  // 如果tag存在，并且是字符串
  if (typeof tag === 'string') {
    let Ctor
    // 当前节点的命名空间，从上下文中获取或者从配置里拿默认值
    ns = (context.$vnode && context.$vnode.ns) || config.getTagNamespace(tag)
    // 判断是否是保留的标签
    if (config.isReservedTag(tag)) {
      // platform built-in elements
      // 创建vnode，拿到命名空间，data, children, undefined, undefined, context
      vnode = new VNode(
        config.parsePlatformTagName(tag), data, children,
        undefined, undefined, context
      )
    } else if ((!data || !data.pre) && isDef(Ctor = resolveAsset(context.$options, 'components', t
g))) {
      // component
      // 从vm实例的option的components中寻找该tag，存在则就是一个组件，创建相应节点，Ctor
组件的构造类
      vnode = createComponent(Ctor, data, context, children, tag)
    } else {
      // unknown or unlisted namespaced elements
      // check at runtime because it may get assigned a namespace when its
      // parent normalizes children
      // 未知的元素，在运行时检查，因为父组件可能在序列化子组件的时候分配一个名字空间
      vnode = new VNode(
        tag, data, children,
        undefined, undefined, context
      )
    }
  } else {
    // direct component options / constructor
    // tag不是字符串的时候，则是组件的构造类
    vnode = createComponent(tag, data, context, children)
```

```
  }
  if (Array.isArray(vnode)) {
    return vnode
  } else if (isDef(vnode)) {
    /*如果有名字空间，则递归所有子节点应用该名字空间*/
    if (isDef(ns)) applyNS(vnode, ns)
    if (isDef(data)) registerDeepBindings(data)
    return vnode
  } else {
    /*如果vnode没有成功创建，则创建空节点*/
    return createEmptyVNode()
  }
}
// 递归算法
// 递归添加
function applyNS (vnode, ns, force) {
  vnode.ns = ns
  if (vnode.tag === 'foreignObject') {
    // use default namespace inside foreignObject
    ns = undefined
    force = true
  }
  if (isDef(vnode.children)) {
    for (let i = 0, l = vnode.children.length; i < l; i++) {
      const child = vnode.children[i]
      if (isDef(child.tag) && (
        isUndef(child.ns) || (isTrue(force) && child.tag !== 'svg'))) {
        applyNS(child, ns, force)
      }
    }
  }
}

// ref #5318
// necessary to ensure parent re-render when deep bindings like :style and
// :class are used on slot nodes
function registerDeepBindings (data) {
  if (isObject(data.style)) {
    traverse(data.style)
  }
  if (isObject(data.class)) {
    traverse(data.class)
  }
}
```

以上代码主要是创建vdom。