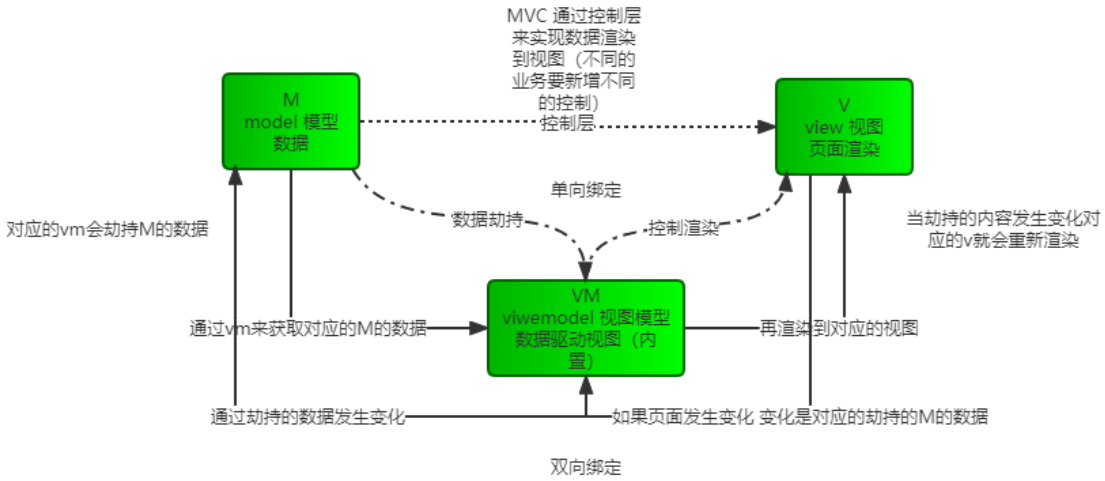
day26 vue相关了解及深拷贝与浅拷贝

VUE

概述:

vue 是一个前端的js库，它简化对应的js原生的操作，提高了对应的浏览器性能（虚拟dom，diff算法）。vue作者（尤雨溪），被阿里巴巴维护（vue2诞生于2015年 vue3诞生于2020年）vue是一个mvvm的框架。



单向绑定

概述:

vm将对应的m的数据 渲染到v上

主要的操作

vm要将m的数据进行劫持， vm再控制v的渲染

vue2的劫持 （es5语法）

Object.defineProperty (对于数组不能进行劫持) + 重写数组的方法（数组内容改变的方法 （7个方法））

vue3的劫持 （es6语法）

proxy (都能劫持 （万物皆对象））

双向绑定

概述:

vm将对于的m的数据渲染到v，v的页面的数据发生变化m也会发生变化

主要的操作

单向相关的操作

vue2的Object.defineProperty

vue3的proxy

监听页面数据发生变化 （ObServer 观察者模式）

vm控制v的渲染

避免大量操作dom （导致多次和重绘和回流 影响性能）

解决方案

虚拟dom （虚拟是一个对象存在于内存中（抽取对应的实体dom形成的对象））

```
<a href=''>哈哈哈<span>嘻嘻<span><a>
//属性文本及子元素 虚拟dom对象
let objA = {
    href:'',
    text:'哈哈哈',
    tagName:'A'
    children:{
        tagName:'SPAN',
        text:'嘻嘻'
    }
}
```

- 相关操作在对应的虚拟dom上进行 （避免重绘回流）
- 完成对应的操作后再进行渲染 （将虚拟dom变成实体dom） （只有一次的重绘和回流操作）

diff算法 （用于比对虚拟dom的差异 （减少渲染量））

模板引擎 （帮助渲染）

双向数据绑定的实现

vue的双向数据绑定

```
<!-- 准备容器 -->
 <div id="app">
   <!-- v-model是vue中实现双向数据绑定的指令 -->
   <input type="text" v-model="message">
   <!-- 查看对应的对应的message的值  模板语法-->
   {{message}}
 </div>
 <!-- 链入vue.js -->
 <script src="./js/vue.js"></script>
 <script>
   new Vue({
     el:'#app' ,//挂载点
     data:{ //数据
       message:'hello world'
     }
   })
 </script>
```

vue2的实现（Object.defineProperty + obServer）

- Object.defineProperty 来进行数据劫持

递归data中的数据来进行劫持

- 当前ObServer监听对应的输入框的内容发生变化
- 重新设置对应的data中的数据（this._data）
- 数据重新设置再进行对应的模板比对渲染对应的页面

```
class Vue {
  constructor(options) {
    let {
      el,
      data
    } = options
    this.el = el //挂载点
    this.data = data //数据
    //用于劫持的容器
    this._data = JSON.parse(JSON.stringify(data))
    //获取对应的内容 el里面内容
    this.content = document.querySelector(this.el)
    //读取里面的 {{}}的内容 进行替换
    this.textContent = this.content.innerHTML
    this.kidnap(this.data, this._data)
    this.comparis()
  }
  //递归遍历data中的数据进行劫持 (递归遍历data中的数据进行劫持操作)
  kidnap(obj, _obj) {
    let that = this
    for (let key in obj) {
      //如果它是一个对象继续往下劫持
      if (typeof obj[key] == 'object') {
        this.kidnap(obj[key], _obj[key])
      }
      //Object.defineProperty来进行劫持
      Object.defineProperty(obj, key, {
        enumerable: true,
        configurable: true,
        get() {
          return _obj[key]
        },
        set(newValue) {
          _obj[key] = newValue
          that.comparis()
        }
      })
    }
  }
  //模板比对解析
  comparis() {
    //{{message}} {message.age.age}
    this.content.innerHTML = this.textContent.replace(/\{\{([\w.]*)\}\}/ig, (v)
=> {
      //{{message}}
      //读取对应的里面key 利用data里面的数据来进行替换
      // console.log(RegExp.$1) //读取分组1里面的内容
      var arr = v.substring(2,v.length-2).split('.')
      let value = this.data
      //遍历获取数据
      for (var i of arr) {
```

```
          value = value[i]
        }
        //遍历arr获取数据
        return value
      })
      //读取对应的input框的v-model属性进行替换
      //先找input框
      let inputs = this.content.querySelectorAll('input')
      let that = this
      //获取所有具备v-model属性的input框架
      Array.from(inputs).filter((v) => {
        return v.getAttribute('v-model')
      }).forEach(v => {
        let key = v.getAttribute('v-model')
        //利用观察者模式进行监听
        //调用obServer
        v.oninput = () => {
          that.data[key] = v.value
        }
        //读取对应的v-model 设置对应的内容
        v.value = that.data[key]
      })
    }

}
```

diff算法比对总结

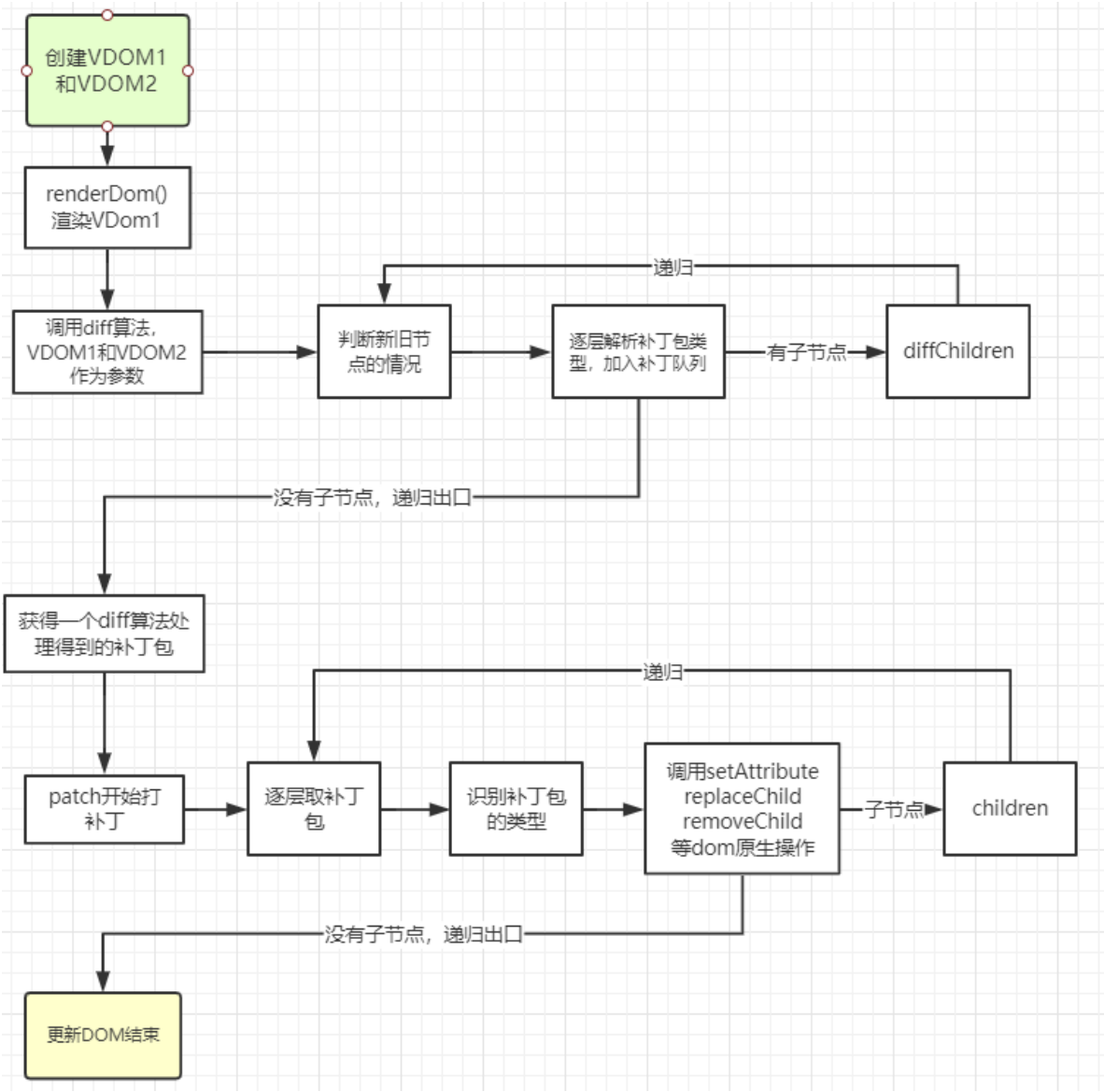
diff算法用于比对新旧虚拟dom，利用打补丁包的形式来比对的。

比对流程

```
//虚拟dom
let vDOM = {
    //虚拟dom中的节点
    vnode:{
        attributes:{},
        text:"hello",
        tagName:'',
        key:'aaaaaxxxsda',
        children:{
            vnode:{
                attributes:{},
                text:"",
                tagName:'',
                key:'dbbbbadba'
            },
            vnode:{
                attributes:{},
                text:"",
                tagName:'',
                key:'ccccddef'
            }
        }
    }
}
```

先比对自身 通过key来找到自身 （key是唯一的 下标不能作为key）

先比对key 再比对自身的属性 比对文本 再比对子元素 递归比到低



深拷贝和浅拷贝

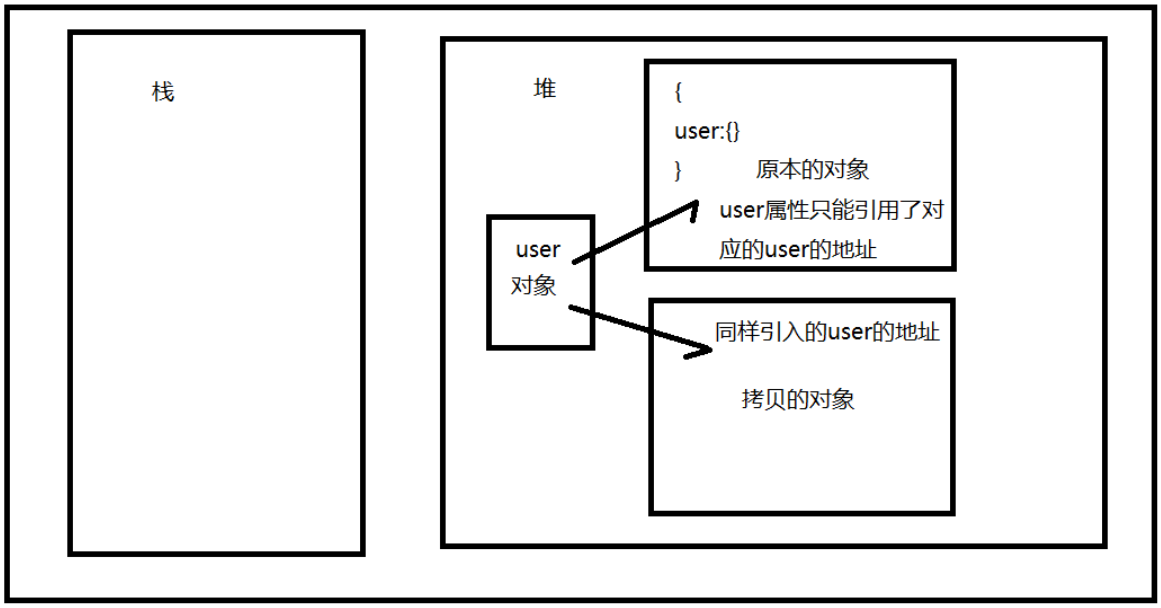
浅拷贝

复制快捷方式 （快捷方式的文件是俩个 但是里面指向的内容都是一个内容）

概述

拷贝的对象和原本的对象不是一个对象，但是内容完全一样，其次里面的内容的地址完全一致

浅拷贝



拷贝第一层值其他全部拷贝地址（所以里面的对象是共用的）

- 浅拷贝不等于赋值它会开辟一个新的内存空间，和原本的地址不一致
- 浅拷贝里面的内容都是拷贝对应的地址 所以和原本的内容地址一致

实现浅拷贝

Object.assign

```
let obj = {
    user: {
        age: 18
    }
}
let newObj = obj
console.log(newObj == obj) //true 赋值的地址是共享的
//浅拷贝
let copyObj = Object.assign({}, obj)
//浅拷贝会产生一个新的对象 和原本的对象地址不一样
console.log(obj == copyObj) //false
//里面的内容的地址是共享的
console.log(obj.user == copyObj.user) //true
obj.user.age = 20
console.log(copyObj.user.age) //20
```

使用扩展运算符实现数组及对象的浅拷贝

```
//使用扩展运算符
let copyObj1 = {...obj}
console.log(copyObj1 == obj) //false
console.log(copyObj1.user == obj.user) //true
let arr = [{age:19},{name:'jack'}]
let copyArr = [...arr]
console.log(copyArr == arr) //false
console.log(copyArr[0] == arr[0]) //true
```

使用数组的concat方法

```
//使用数组的concat方法实现数组的浅拷贝
let concatArr = [].concat(arr)
console.log(concatArr == arr) //false
console.log(concatArr[0] == arr[0]) //true
```

使用数组的slice方法

```
//使用数组的slice方法
let sliceArr = arr.slice()
console.log(sliceArr == arr) //false
console.log(sliceArr[0] == arr[0]) //true
```

使用自定义函数遍历

```
function clone(obj){
    let newObj = {}
    for(let key in obj){
        newObj[key] = obj[key]
    }
    return newObj
}
let obj = {user:{}}
let copyObj = clone(obj)
console.log(copyObj == obj) //false
console.log(copyObj.user == obj.user) //true
```

第三方插件 lodash.js (提供的clone方法)

https://www.lodashjs.com/

```
let obj1 = {
    user: {}
}
let cloneObj = _.clone(obj1)
console.log(cloneObj == obj1)//false
console.log(cloneObj.user == obj1.user)//true
```

深拷贝

文件复制粘贴

概述

拷贝的是对应的值，不拷贝地址。

实现方式

JSON.Stringify JSON.parse

```
let obj = {list:['1','2'],user:{name:'tom'}}
let copyObj = JSON.parse(JSON.stringify(obj))
console.log(obj == copyObj) //false
console.log(obj.list == copyObj.list) //false
console.log(obj.user == copyObj.user) //false
console.log(obj.user.name == copyObj.user.name) //true
```

使用lodash.js cloneDeep方法

```
//使用lodash.js _.cloneDeep
let cloneObj =  _.cloneDeep(obj);
console.log(obj == cloneObj) //false
console.log(obj.list == cloneObj.list) //false
console.log(obj.user == cloneObj.user) //false
console.log(obj.user.name == cloneObj.user.name) //true
```

自定义递归书写对应的深拷贝 （重点）

```
//自定义函数实现 （递归）
//自定义方法传入对应的需要拷贝的对象
function deepClone(obj) {
    //判断是否为对象 如果为对象那么进行拷贝 如果不是对象直接返回
    //如果不是对象 或者 它为null
    if(typeof obj == 'function'){
       //返回新的函数
       return obj.bind(this)
    }
    if (typeof obj != 'object' || !obj) {
        //直接返回
        return obj
    }
    //如果是对象 Regexp Object Array Date
    if (obj instanceof RegExp) { //如果是正则
        return new RegExp(obj)
    }
    //如果它是Date 日期
    if (obj instanceof Date) {
        return new Date(obj.getTime())
    }
    let copyObj = {}
    //判断是数组还是对象
    if (obj instanceof Array) {
        copyObj = []
    }
    //遍历对应的对象里面内容
    for (let key in obj) {
        //递归
        copyObj[key] = deepClone(obj[key])
    }
    return copyObj
}
```

总结

- vue是一个mvvm的框架，vm是内置的，不需要你去管理。
- vue的数据劫持是通过 Object.defineProperty (vue2) 重写了数组的7个方法（不能对于数组进行劫持） Proxy (vue3)
- vue主要劫持是data中的数据 （_data的属性）递归去进行劫持
- vue的双向数据绑定主要是通过数据劫持+obServer（观察者模式）
- vue是利用虚拟dom来进行对应的比对（里面采用diff算法）使用模板引擎进行解析渲染
- diff算法比对先比对自身（key），再比对对应的vnode，再比对对应的子节点（递归比对） 采用patch补丁包的形式来进行重新渲染

- 深拷贝和浅拷贝 深拷贝拷贝的是值 浅拷贝拷贝的是地址 不管深浅拷贝都会产生一个新的对象