**Vue源码**

<template>模板用于保客户端内容的机制

用div包住才会发请求，没有则是**文档碎片(document-fragment)**没有父级元素的最小文档对象，和dom树没有作用，所以一起append都放里面 就不可以提高性能，想做什么就做什么，不会有性能问题，操作完之后，再放回dom树中

拿到文档碎片template.content.childNodes

**将伪数组转成真数组**

Arrary.apply(null,伪数组)

Arrary.prototypes.slice.call(伪数组)

MVVM

**数据代理（vue）**

1. 引入vue.js
2. new Vue({

data:{

msg:”msg”

}

})

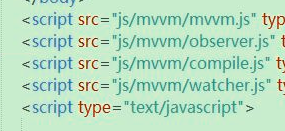
console.log(vm.msg)

vm和vm\_data不是指向同一个引用，但是对他们的属性进行读写操作却是同步的

这种模式叫做**数据代理**，不相等，但操作相等

MVVM

1. 引入MVVM.js



实现了数据代理

1. 将配置对象给了一个叫做$options的属性，这个属性挂靠在了vm身上
2. 将options的data属性挂靠给vm的\_data属性

vm.data和配对象中的data指向同一个

1. 将配置对象中的data对象对应的所有属性，将属性转成访问描述符绑在vm身上
2. 由于vm身上的所有属性都是访问描述符，在做获取用get，从data身上去拿数据，又由于\_data和配置对象中的data指向同一个引用，进而实现了vm对vm\_data的数据代理
3. Object.keys（data） 返回可枚举属性的数组

set：

**{{}}数据绑定**

1. 根据配置对象的el属性去查找挂靠节点
2. 将挂靠节点中的所有子节点剪切到一个文档碎片
3. 解析{{}}，从配置对象的data中寻找对应的属性值，拿到属性值替换这个模板

1).调用init()方法

2).this.complieElement(this.fragment)调用文档碎片

3) 将文档碎片中的子节点全部拿出来，将他们转成真数组去循环

4).拿到每个节点的信息，判断是元素节点（me.complie(node)），还是包含{{}}大括号的文本加点（me.complieText）

5).调用工具类的text函数处理文本节点 compileUtil.text

6).基础函数bind this.bind(node, vm, exp, 'text');

4. \_getVMVal !!!!! 从配置对象得data中寻找对应得属性值

1).调用对应的更新器

2). 调用对应的更新器

textUpdater: function(node, value) {

node.textContent = typeof value == 'undefined' ? '' : value; }

**普通指令解析**

**:class和v-on解析**

**数据流：**

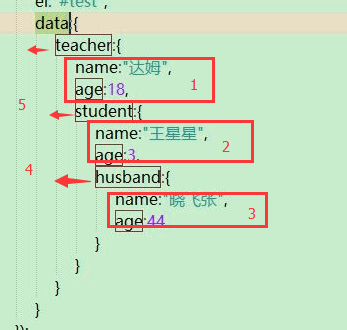
**数据劫持：给配置对象中的data对象的所有属性将数据描述符改成了访问描述符**

**进入observe，通过mvvm做data数据劫持，发现是对象，进入构造，进入data**

**调用work方法，遍历data，一层一层拨开，key是第一个对象，data[key]对应的值，**

**又回到observe，做到一半发现时是对像劫持，，又回到下一个不是对象的**

**先做基本数据类型做访问描述符，再将对象做成访问描述符**

**先把基本数据类型转成访问描述符。再把对象的属性描述符变成访问描述符**

**数据流：**

**指令解析的最后一步，构造函数watcher（订阅者），让订阅者主动联系发布者dep，**

**建立多对多的订阅发布模型，**

根据表达式，dep为属性值创造的，每个属性都有一个id，通过watch来更新的，多对多的关系，用id放进去watch

new watch只要是指令在解析，在解析过程中可以拿到属性的dep，吧dep塞到watch对象里面，两者的建立是在指令解析的时候建立的，指令一个解析一个，有变化就通知对应的watch去改数据

**调用get去通知dep，和dep通过，dep.target来联系，把watch给dep。dep里面有数组，放watch，回来自己里面放，**

**单向数据流：**

**数据劫持后，进入get的方法**