链滴

# 从 Vue 源码分析 Vue 执行过程

作者：gmw-zjw

原文链接：https://ld246.com/article/1571053934852

来源网站：链滴

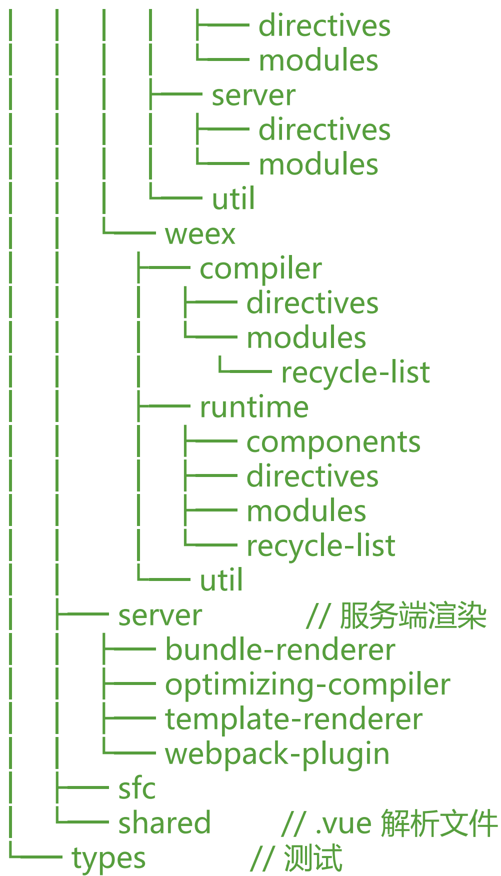
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

初入茅庐，开始我们的vue之旅！

刚开始看源码，不要一行一行看，建议新建一 demo debug慢慢调试，一点一点读，这里采用配合目的形式，加深我们在项目中对具体方法的认知！

打开vue源码项目你会发现有这么几个目录，如下：

```
# bash
.
├── dist            // dist
├── src               // 源码目录
│   ├── compiler       // 编译相关
│   │   ├── codegen
│   │   ├── directives
│   │   └── parser
│   ├── core            // 核心
│   │   ├── components
│   │   ├── global-api
│   │   ├── instance
│   │   │   └── render-helpers
│   │   ├── observer
│   │   ├── util
│   │   └── vdom
│   │      ├── helpers
│   │      └── modules
│   ├── platforms       // 平台相关
│   │   ├── web
│   │   │   ├── compiler
│   │   │   │   ├── directives
│   │   │   │   └── modules
│   │   │   ├── runtime
│   │   │   │   ├── components
```




compiler: 主要用于编译

core: 这里是核心中的核心，主要是vnode、核心api、等一些重要的封装

platforms: 适配多端平台， 比如 native（vexx）、web

server: 服务端渲染

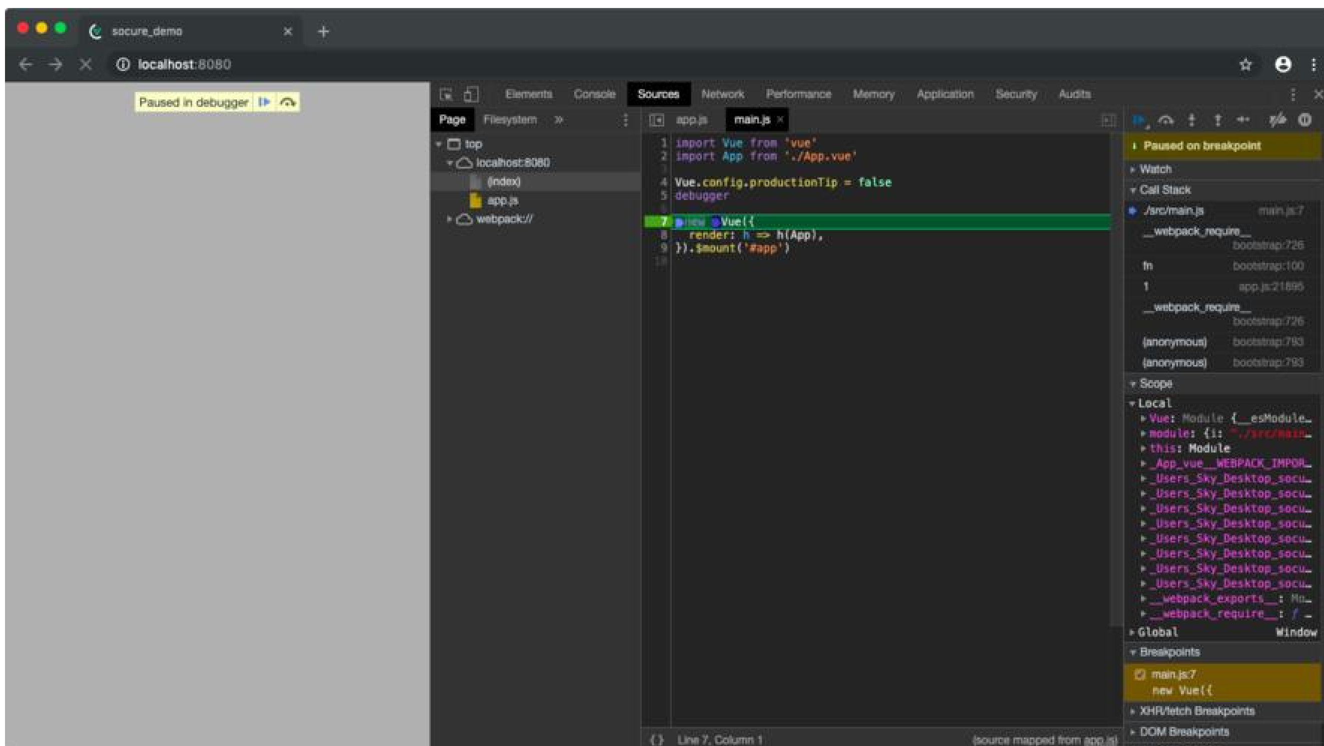
sfc: 这里主要是webpack对vue代码编译相关

shared: 对 .vue 文件的解析

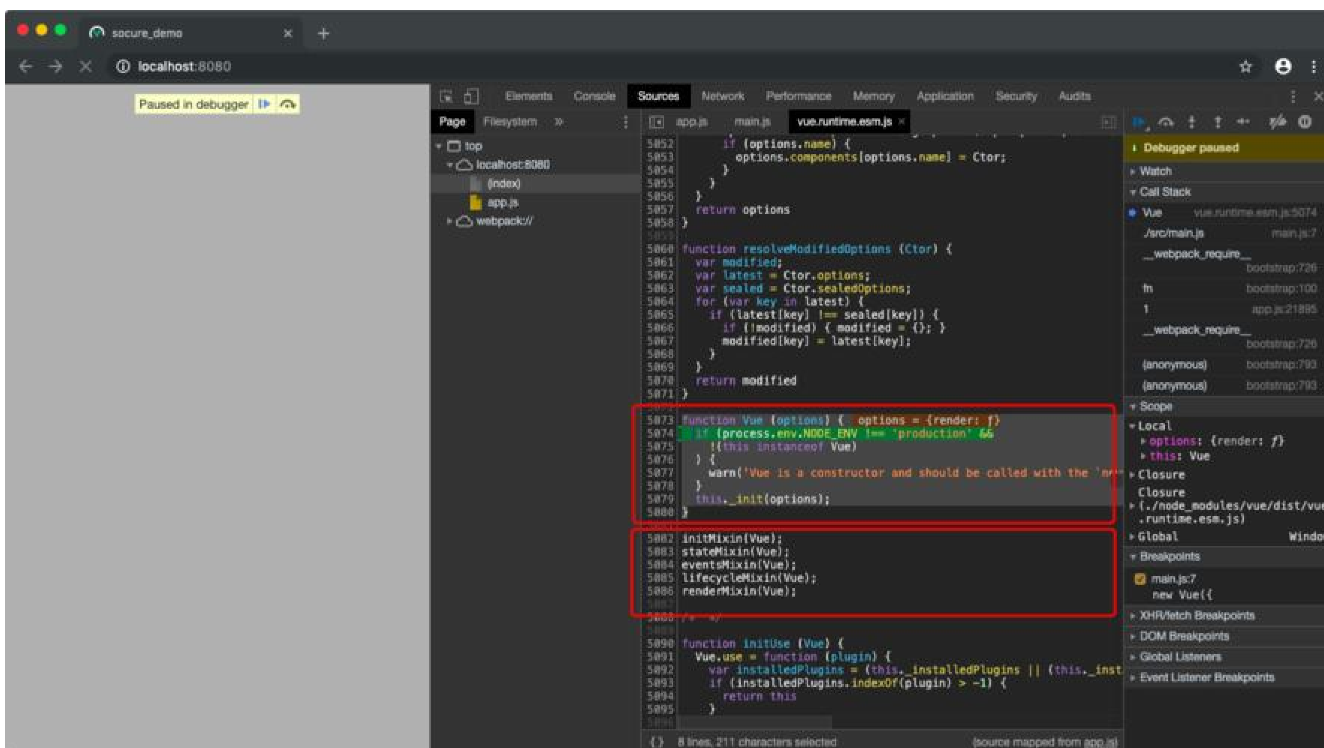
## 一、入口文件加载

在我们配置好项目后，debugger，点击进入：

原文链接：从 Vue 源码分析 Vue 执行过程




进入断点：




会看到这样两段代码，这里定义了一个Vue实例，也就是new Vue的出处，

这里执行了 this._init(options)，进而我们看看这个方法做了那些事，

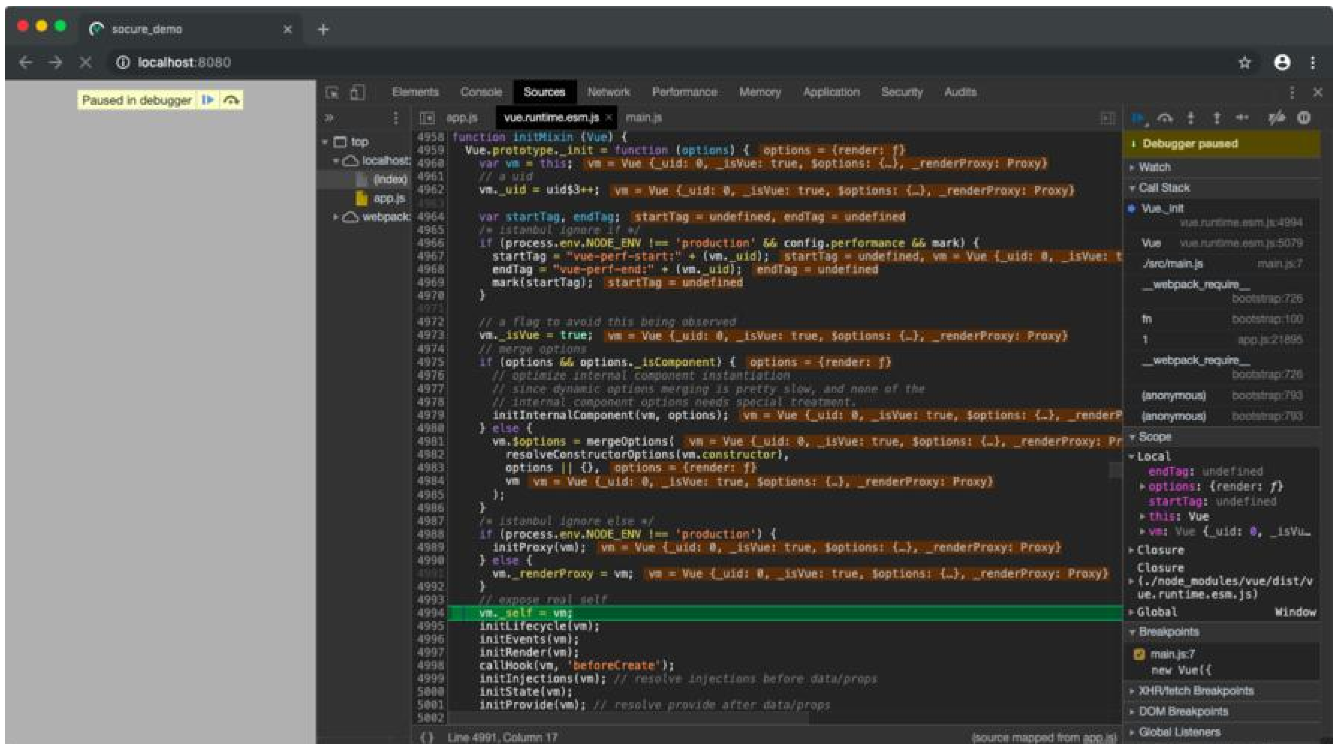
这里直接贴上源码，方便阅读理解：

```
// 混入方法，接收Vue实例<Component>
export function initMixin (Vue: Class<Component>) {
  Vue.prototype._init = function (options?: Object) {
    const vm: Component = this
    // a uid
```

```
  // 唯一递增的id
  vm._uid = uid++

  let startTag, endTag
  /* istanbul ignore if */
  if (process.env.NODE_ENV !== 'production' && config.performance && mark) {
    startTag = `vue-perf-start:${vm._uid}`
    endTag = `vue-perf-end:${vm._uid}`
    mark(startTag)
  }

  // a flag to avoid this being observed
  // 如果是vue实例，则不需要被observe
  vm._isVue = true
  // merge options
  // 第一步： 对options参数进行处理
  if (options && options._isComponent) {
    // optimize internal component instantiation
    // since dynamic options merging is pretty slow, and none of the
    // internal component options needs special treatment.
    initInternalComponent(vm, options)
  } else {
    vm.$options = mergeOptions(
      resolveConstructorOptions(vm.constructor),
      options || {},
      vm
    )
  }
  /* istanbul ignore else */
  // 第二步 renderProxy
  if (process.env.NODE_ENV !== 'production') {
    initProxy(vm)
  } else {
    vm._renderProxy = vm
  }
  // expose real self
  vm._self = vm
  // 第三步 vm生命周期钩子
  initLifecycle(vm)// 做了一些生命周期的初始化工作，初始化了很多变量，最主要是设置了父子组
的引用关系，也就是设置了 `$parent` 和 `$children`的值

  // 第四步 vm 事件初始化监听
  initEvents(vm)// 注册事件，注意这里注册的不是自己的，而是父组件的。因为很明显父组件的监
器才会注册到孩子身上。
  initRender(vm)// 做一些 render 的准备工作，比如处理父子继承关系等，并没有真的开始 render
  callHook(vm, 'beforeCreate')// 准备工作完成，接下来进入 `create` 阶段
  initInjections(vm) // resolve injections before data/props

  // 第五步 vm状态初始化
  initState(vm)// `data`, `props`, `computed` 等都是在这里初始化的，常见的面试考点比如`Vue
如何实现数据响应化的` 答案就在这个函数中寻找
  initProvide(vm) // resolve provide after data/props
  callHook(vm, 'created')
```

```
    /* istanbul ignore if */
    if (process.env.NODE_ENV !== 'production' && config.performance && mark) {
      vm._name = formatComponentName(vm, false)
      mark(endTag)
      measure(`vue ${vm._name} init`, startTag, endTag)
    }

    // 第六步 render 挂在到 $mount
    if (vm.$options.el) {
      vm.$mount(vm.$options.el) // el $mount
    }
  }
}
```

debugger过程图示：




## options:

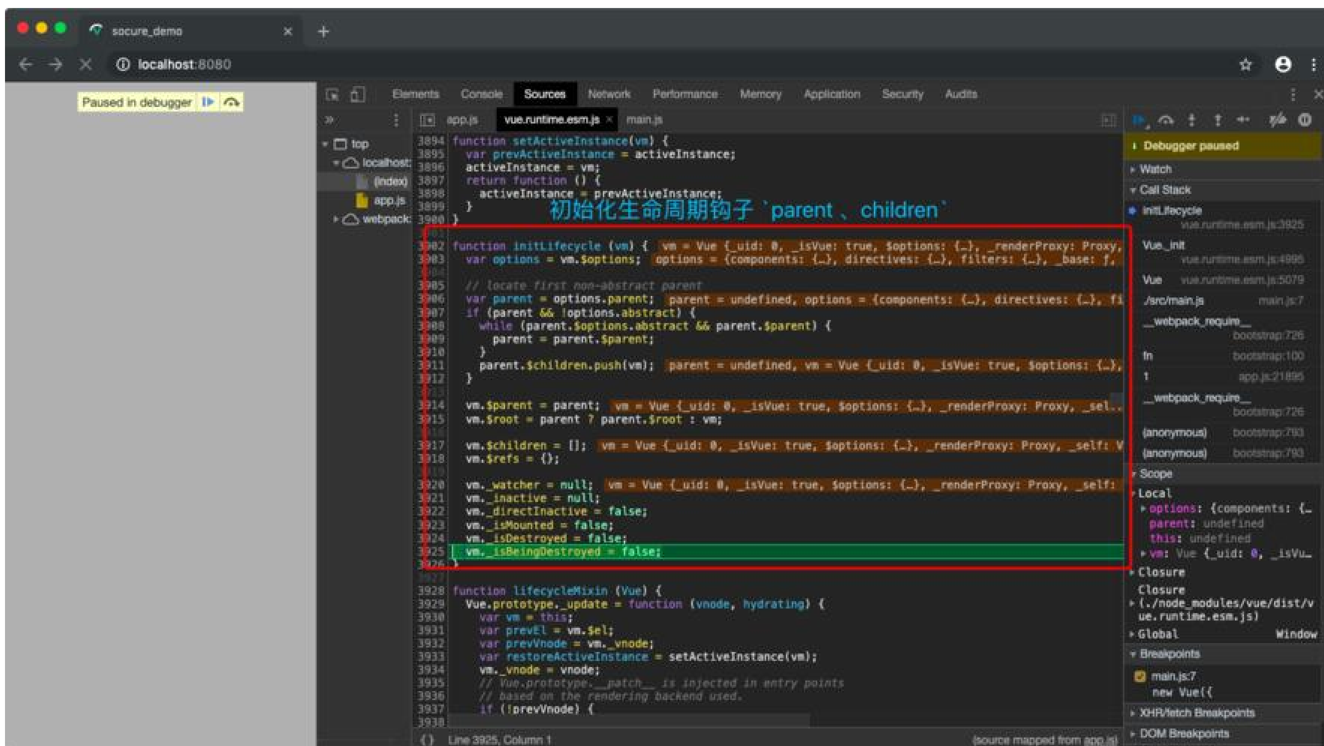
首先vue会先对options参数进行处理，这里主要是合并options参数，具体下面讲.

## renderProxy:

这里使用Proxy代理，把 render 中的this指向 renderProxy,也就是vm实例。

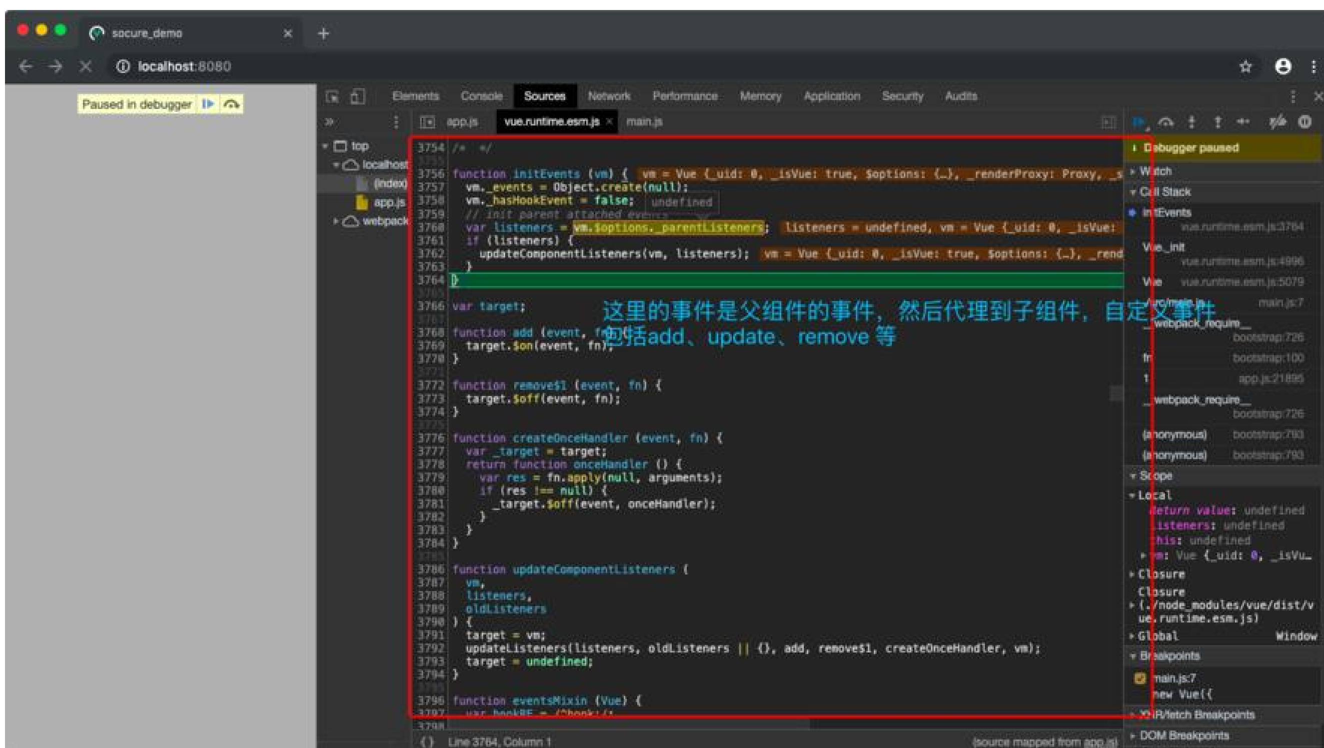
## initLifecycle:

这里做了一些生命周期的初始化工作，初始化了很多变量，最主要是设置了父子组件的引用关系，也是设置了 $parent 和 $children的值。




## initEvents:

注册事件，注意这里注册的不是自己的，而是父组件的。因为父组件的监听器才会注册到子组件身上。




源码：

```
//instance/events.js
/*
 * @Description: In User Settings Edit
 * @Author: your name
 * @Date: 2019-08-14 10:42:22
```

```
 * @LastEditTime: 2019-08-15 16:41:19
 * @LastEditors: Please set LastEditors
 */
/* @flow */

import {
  tip,
  toArray,
  hyphenate,
  formatComponentName,
  invokeWithErrorHandling
} from '../util/index'
import { updateListeners } from '../vdom/helpers/index'

// 初始化event事件
export function initEvents (vm: Component) {
  // 创建一个新对象，并复制给vm._events实例
  vm._events = Object.create(null)
  // bool钩子用于标志位作用，这里没用哈希表查找是否有钩子，可以减少开销、性能优化
  vm._hasHookEvent = false
  // init parent attached events
  // 初始化父组件attached的事件
  const listeners = vm.$options._parentListeners
  if (listeners) {
    updateComponentListeners(vm, listeners)
  }
}

let target: any

function add (event, fn) {
  target.$on(event, fn)
}

function remove (event, fn) {
  target.$off(event, fn)
}

function createOnceHandler (event, fn) {
  const _target = target
  return function onceHandler () {
    const res = fn.apply(null, arguments)
    if (res !== null) {
      _target.$off(event, onceHandler)
    }
  }
}

export function updateComponentListeners (
  vm: Component,
  listeners: Object,
  oldListeners: ?Object
) {
  target = vm
```

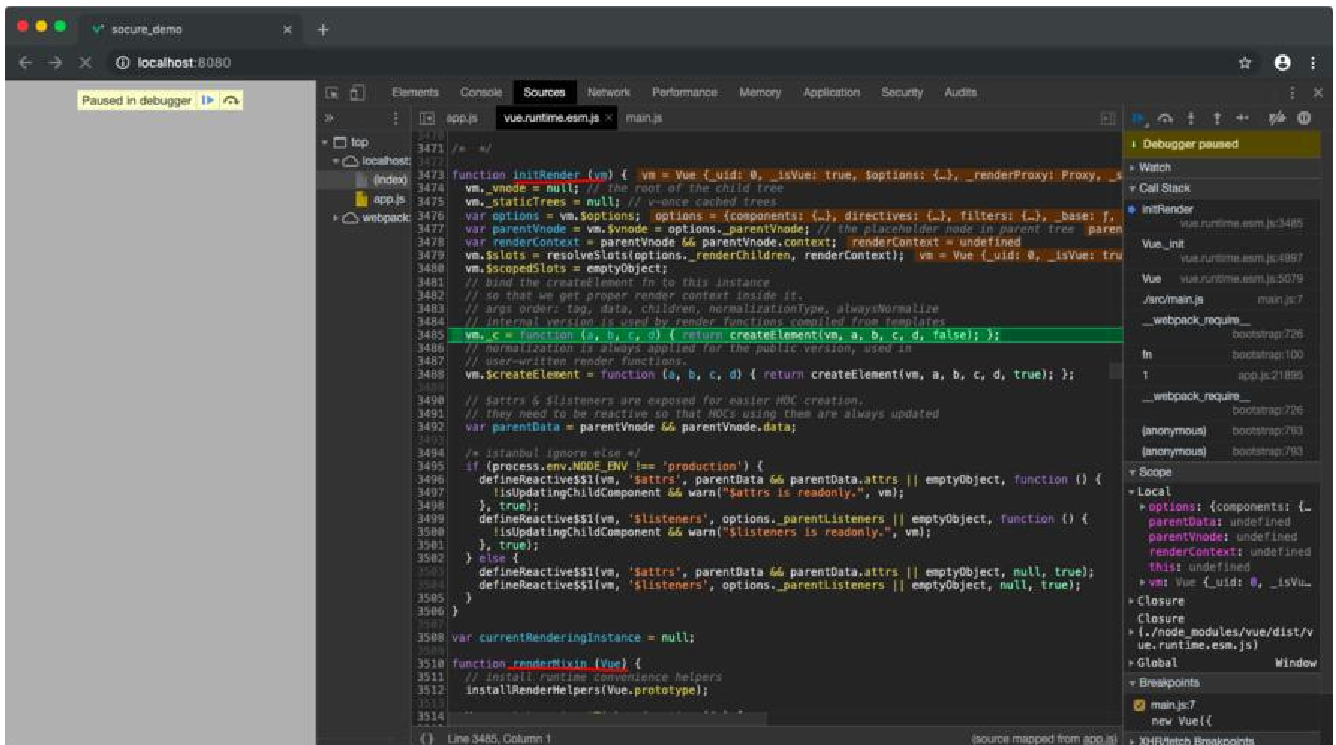
```
  updateListeners(listeners, oldListeners || {}, add, remove, createOnceHandler, vm)
  target = undefined
}

export function eventsMixin (Vue: Class<Component>) {
  const hookRE = /^hook:/
  /**
   * 绑定一个自定义监听事件
   * @event {String|Array<string>} event
   * @fn {function} function
   * @returns Component
   */
  Vue.prototype.$on = function (event: string | Array<string>, fn: Function): Component {
    const vm: Component = this
    // 如果是数组，则递归为每个绑定$on
    if (Array.isArray(event)) {
      for (let i = 0, l = event.length; i < l; i++) {
        vm.$on(event[i], fn)
      }
    } else {
      (vm._events[event] || (vm._events[event] = [])).push(fn)
      // optimize hook:event cost by using a boolean flag marked at registration
      // instead of a hash lookup
      // 改变标识位，这里用bool是为了性能优化
      if (hookRE.test(event)) {
        vm._hasHookEvent = true
      }
    }
    return vm
  }

  /**
   * 监听一个只能监听一次的事件，用完直接被销毁
   * @event {String} 事件
   * @fn {function} fn 回调函数
   * @returns Component
   */
  Vue.prototype.$once = function (event: string, fn: Function): Component {
    const vm: Component = this
    function on () {
      // 在第一执行后直接移除
      vm.$off(event, on)
      // 执行注册方法
      fn.apply(vm, arguments)
    }
    on.fn = fn
    vm.$on(event, on)
    return vm
  }

  /**
   * 销毁事件
   * @event {String|Array<string>} event
   * @fn {function} function
```

```
 * @returns Component
 */
Vue.prototype.$off = function (event?: string | Array<string>, fn?: Function): Component {
  const vm: Component = this
  // all
  // 如果不传参直接删除所有事件
  if (!arguments.length) {
    vm._events = Object.create(null)
    return vm
  }
  // array of events
  // 如果传入参数是数组，则递归删除
  if (Array.isArray(event)) {
    for (let i = 0, l = event.length; i < l; i++) {
      vm.$off(event[i], fn)
    }
    return vm
  }
  // specific event
  const cbs = vm._events[event]
  // 本身不存在直接返回
  if (!cbs) {
    return vm
  }
  if (!fn) {
    vm._events[event] = null
    return vm
  }
  // specific handler
  // 遍历寻找对应方法并删除
  let cb
  let i = cbs.length
  while (i--) {
    cb = cbs[i]
    if (cb === fn || cb.fn === fn) {
      cbs.splice(i, 1)
      break
    }
  }
  return vm
}

/**
 * 触发当前实例上的事件。附加参数都会传给监听器回调
 * @event {String} event
 * @returns Component
 */
Vue.prototype.$emit = function (event: string): Component {
  const vm: Component = this
  // 如果是生产环境 dev
  if (process.env.NODE_ENV !== 'production') {
    const lowerCaseEvent = event.toLowerCase()
    if (lowerCaseEvent !== event && vm._events[lowerCaseEvent]) {
      tip(
```

```
          `Event "${lowerCaseEvent}" is emitted in component ` +
          `${formatComponentName(vm)} but the handler is registered for "${event}". ` +
          `Note that HTML attributes are case-insensitive and you cannot use ` +
          `v-on to listen to camelCase events when using in-DOM templates. ` +
          `You should probably use "${hyphenate(event)}" instead of "${event}".`
        )
      }
    }
    let cbs = vm._events[event]
    if (cbs) {
      // 将类数组转成数组
      cbs = cbs.length > 1 ? toArray(cbs) : cbs
      const args = toArray(arguments, 1)
      const info = `event handler for "${event}"`
      // 遍历执行
      for (let i = 0, l = cbs.length; i < l; i++) {
        invokeWithErrorHandling(cbs[i], vm, args, vm, info)
      }
    }
    return vm
  }
}
```

## initRender:

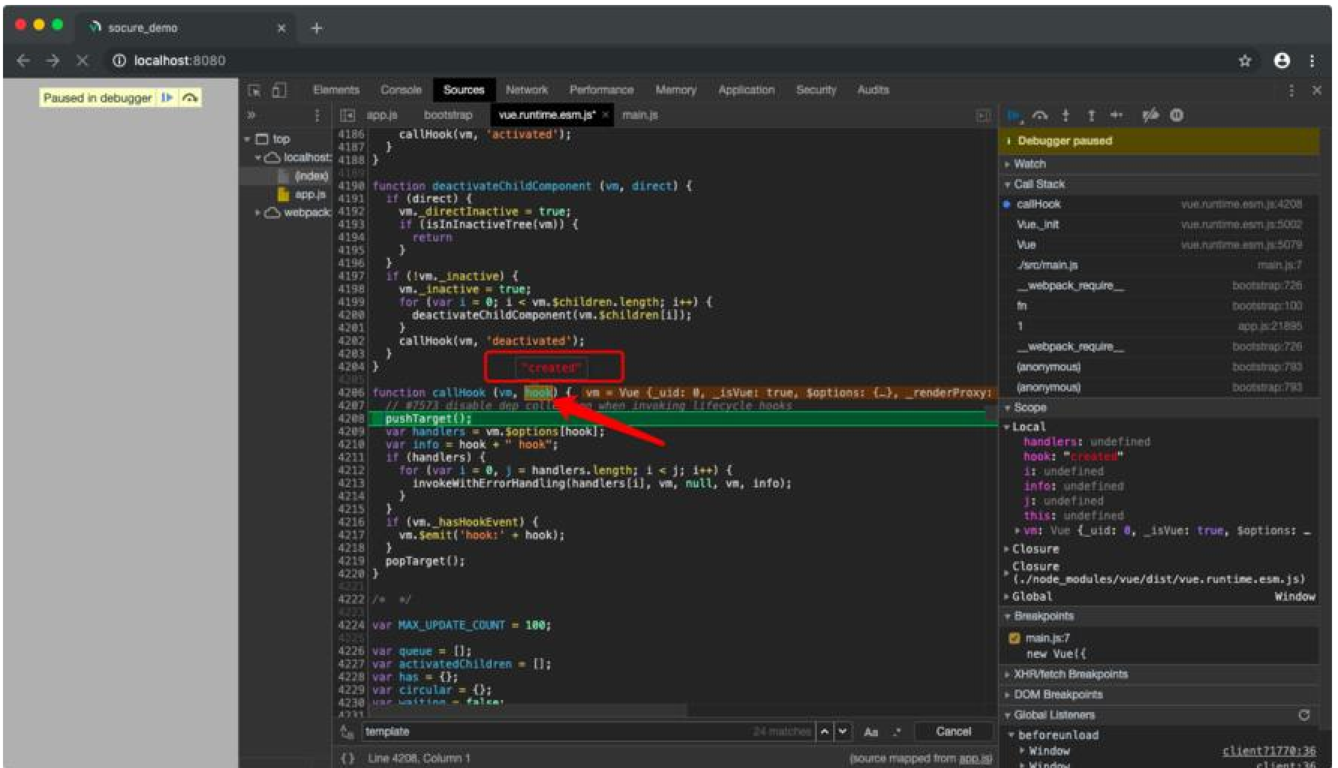
做一些 render 的准备工作，比如处理父子继承关系等，并没有真的开始 render。




## calHook:

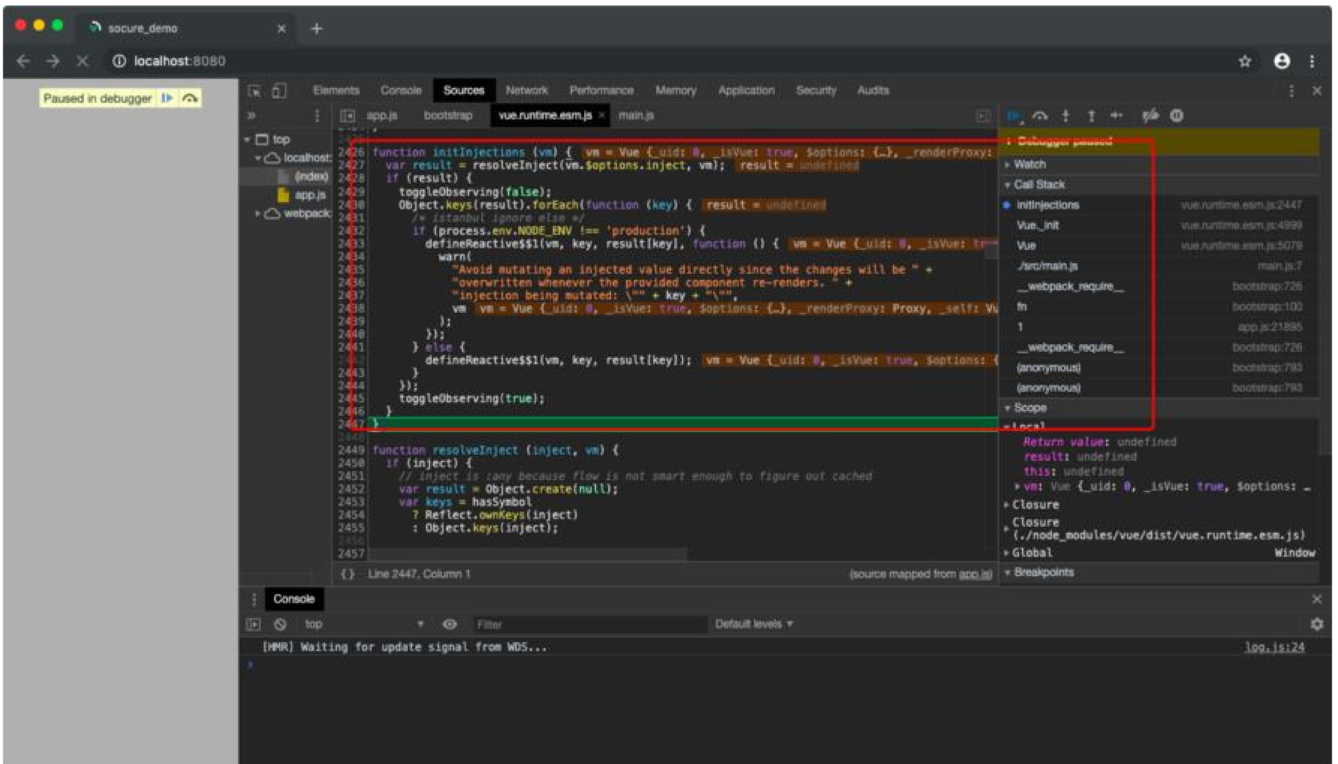
准备工作完成，接下来进入 create 阶段。

原文链接：从 Vue 源码分析 Vue 执行过程



## initInjections:

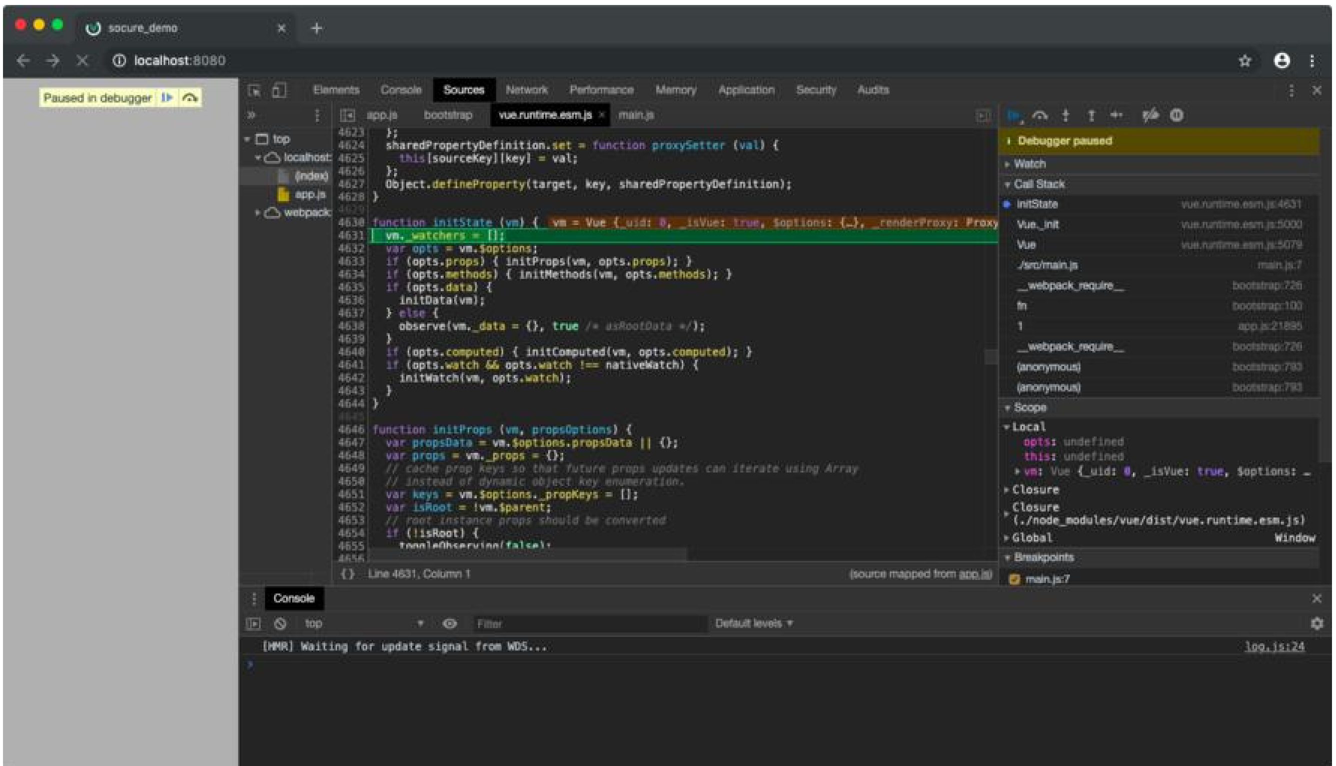
初始化 data、props数据



## initState:

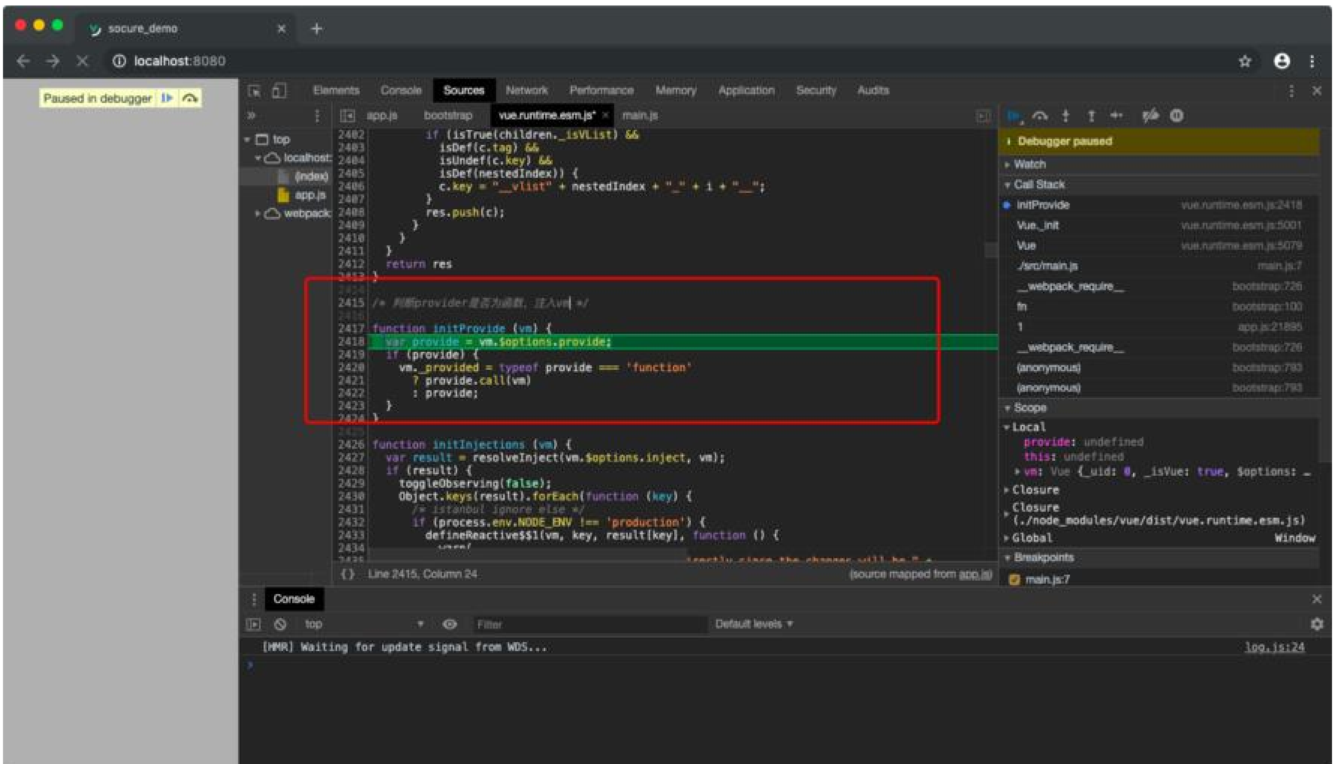
初始化 data、props、computed、等

原文链接：[从 Vue 源码分析 Vue 执行过程]()



## initProvider:

加载provider

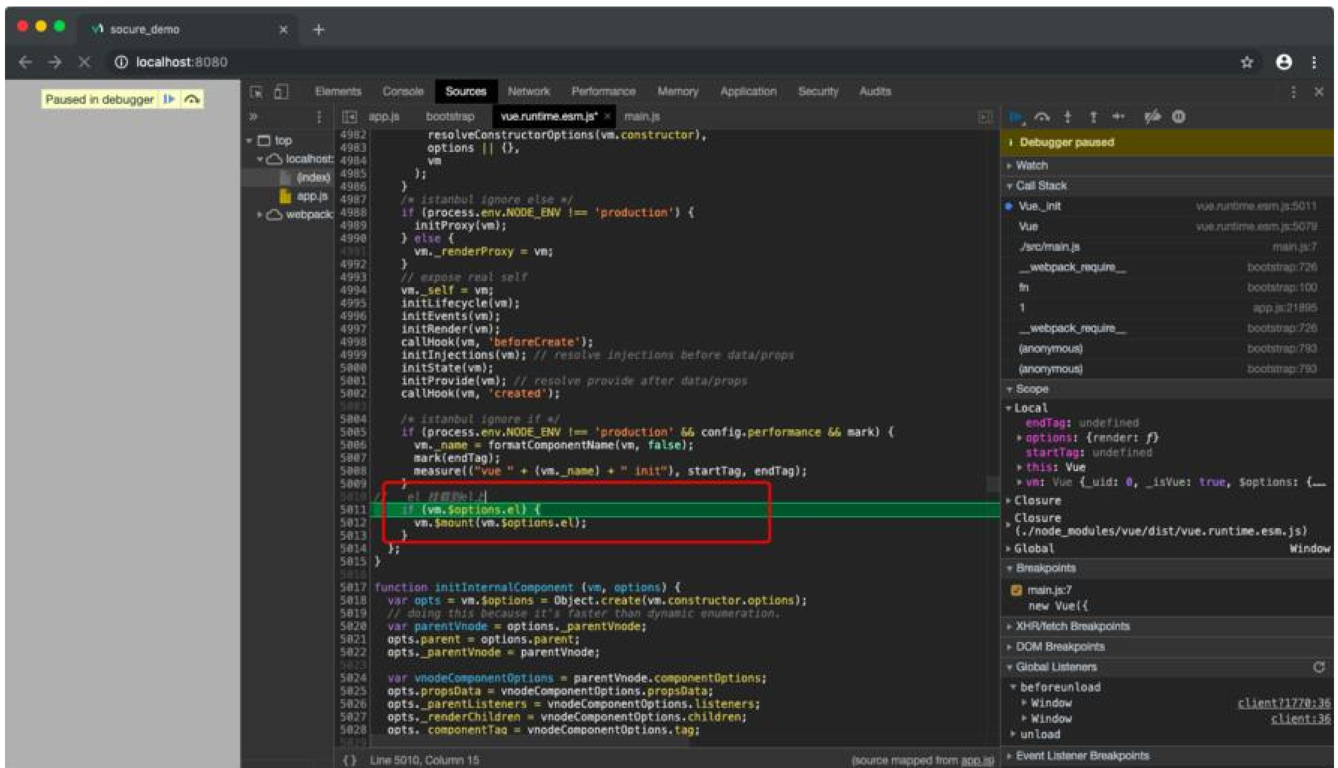


最后就是将render挂在到$mount上

```
// 挂载$mount到el上
if (vm.$options.el) {
    vm.$mount(vm.$options.el)// el $mount
```

原文链接：从 Vue 源码分析 Vue 执行过程

```
}
```




接下来解释每个的主要功能！

在 initState 中做了以下几件事:

```
function initState (vm) {
  vm._watchers = [];
  var opts = vm.$options;
//   初始化 props
  if (opts.props) { initProps(vm, opts.props); }
//   初始化 methods
  if (opts.methods) { initMethods(vm, opts.methods); }
  if (opts.data) {
//     初始化data
    initData(vm);
  } else {
//     没有data是设置data为一个 {}
    observe(vm._data = {}, true /* asRootData */);
  }
//   初始化 computed
  if (opts.computed) { initComputed(vm, opts.computed); }
//   初始化 watch
  if (opts.watch && opts.watch !== nativeWatch) {
    initWatch(vm, opts.watch);
  }
}
```

initData:

```
function initData (vm) {
  // 拿到data
```

原文链接：从 Vue 源码分析 Vue 执行过程

```
  var data = vm.$options.data;
  // 这里就是为什么我们在跟组件要写data:() 而在其他位置要写成函数的原因
  data = vm._data = typeof data === 'function'
    ? getData(data, vm)
    : data || {};
//     判断data是否为Object
  if (!isPlainObject(data)) {
    data = {};
    process.env.NODE_ENV !== 'production' && warn(
      'data functions should return an object:\n' +
      'https://vuejs.org/v2/guide/components.html#data-Must-Be-a-Function',
      vm
    );
  }
  // proxy data on instance
  var keys = Object.keys(data);
  var props = vm.$options.props;
  var methods = vm.$options.methods;
  var i = keys.length;
//   遍历
  while (i--) {
    var key = keys[i];
    if (process.env.NODE_ENV !== 'production') {
      if (methods && hasOwn(methods, key)) {
        warn(
          ("Method \"" + key + "\" has already been defined as a data property."),
          vm
        );
      }
    }
    if (props && hasOwn(props, key)) {
      process.env.NODE_ENV !== 'production' && warn(
        "The data property \"" + key + "\" is already declared as a prop. " +
        "Use prop default value instead.",
        vm
      );
    } else if (!isReserved(key)) {
//       使用代理将data代理到vm实例上
      proxy(vm, "_data", key);
    }
  }
  // observe data
//  使用observe的方式definProperty数据双向绑定，Object.defineProperty通过setter设置值，通getter获取值（依赖收集）
  observe(data, true /* asRootData */);
}
```

使用Observer中definProperty对数据进行双向绑定，Object.definPropert实现了getter、setter 等使用getter进行依赖收集（Dep），setter 设置data 中的值。

这里是Dep依赖收集