VUE

Substituição

de

Variáveis

VUE – Substituição de variáveis

1. Vamos mostrar o processo de substituição das variáveis VUE desde a inicialização do objeto VUE, passando pelo processo de compilação de uma expressão textual, até a substituição propriamente dita. O código está abaixo:

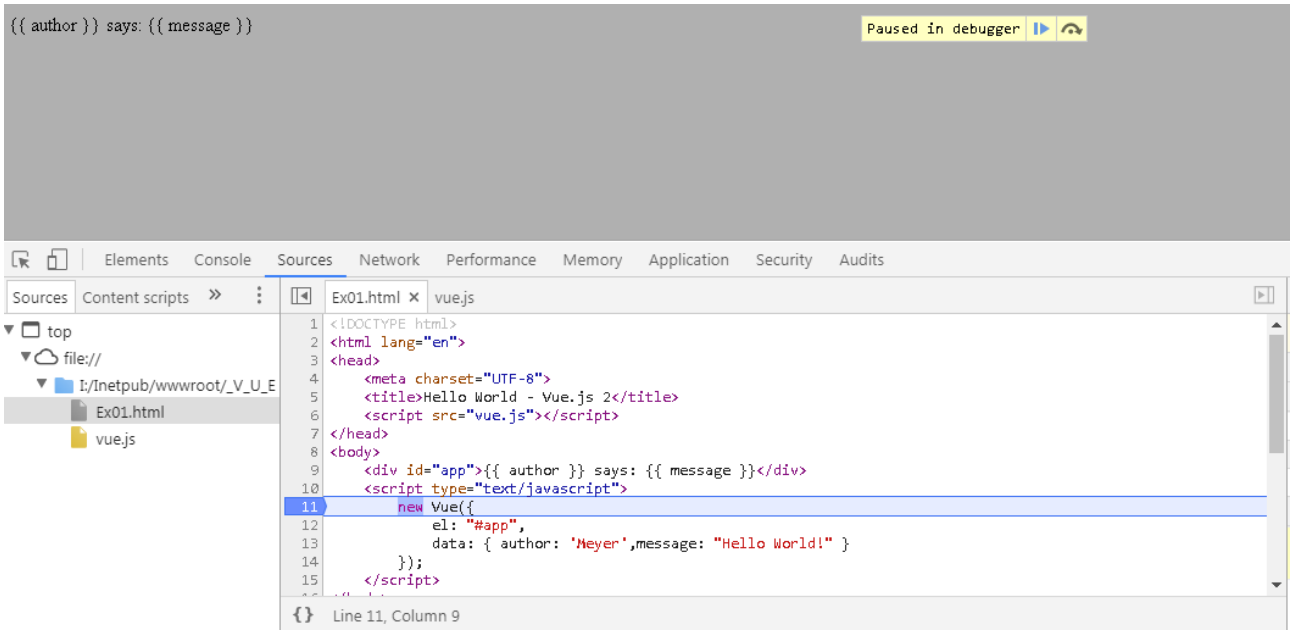


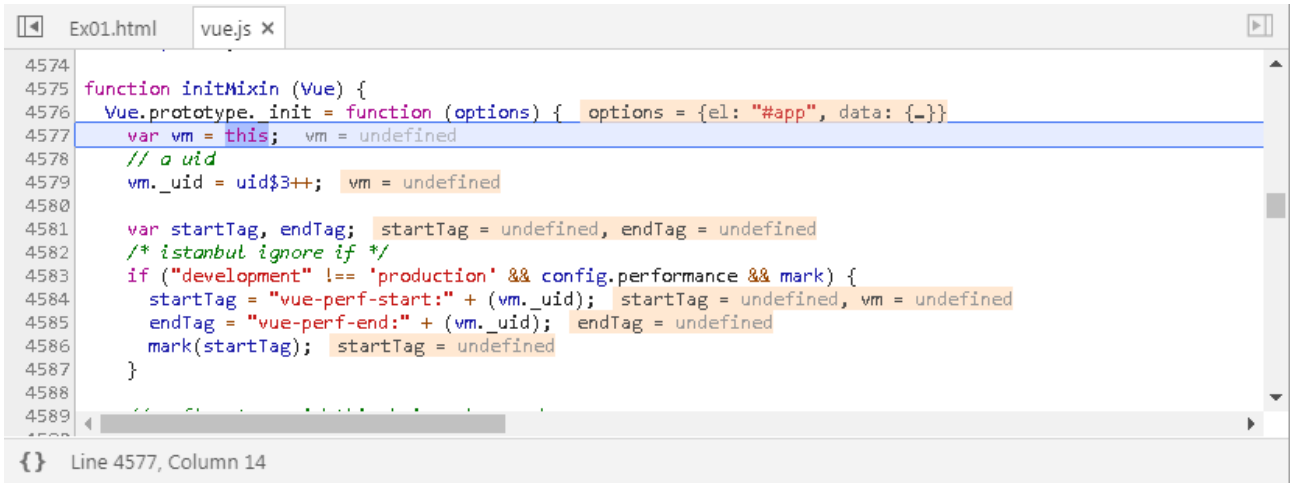
Figura 1: A parte cinza do debug mostra a exibição no formato de substituição, ainda sem os valores substituídos.

2. Quando um objeto Vue é instanciado, é chamada a função Vue da biblioteca Vue.js, com as opções como parâmetro:

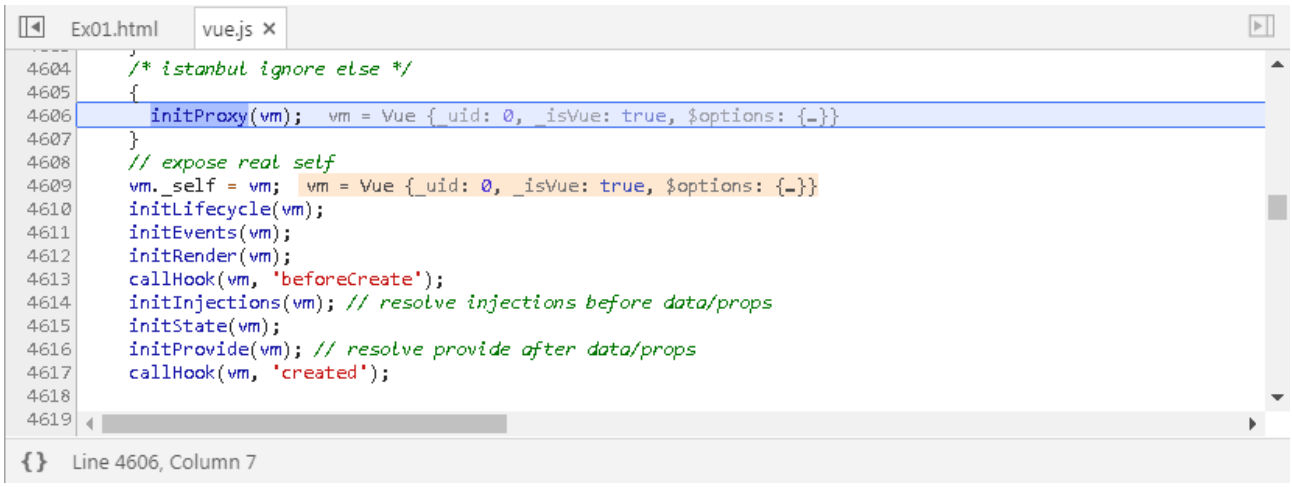
Figura 2: Não é preciso dar muita importâmcia ao teste de ambiente (development ou prodution).

VUE – Substituição de variáveis

3. O que parece uma função, **this._init** da figura anterior, é realmente a definição de um protótipo de objeto que reside na função **initMixin**, que recebe como base a instância do objeto **Vue**. O conteúdo das opções está mostrado no debuf, ao final da linha 4576.



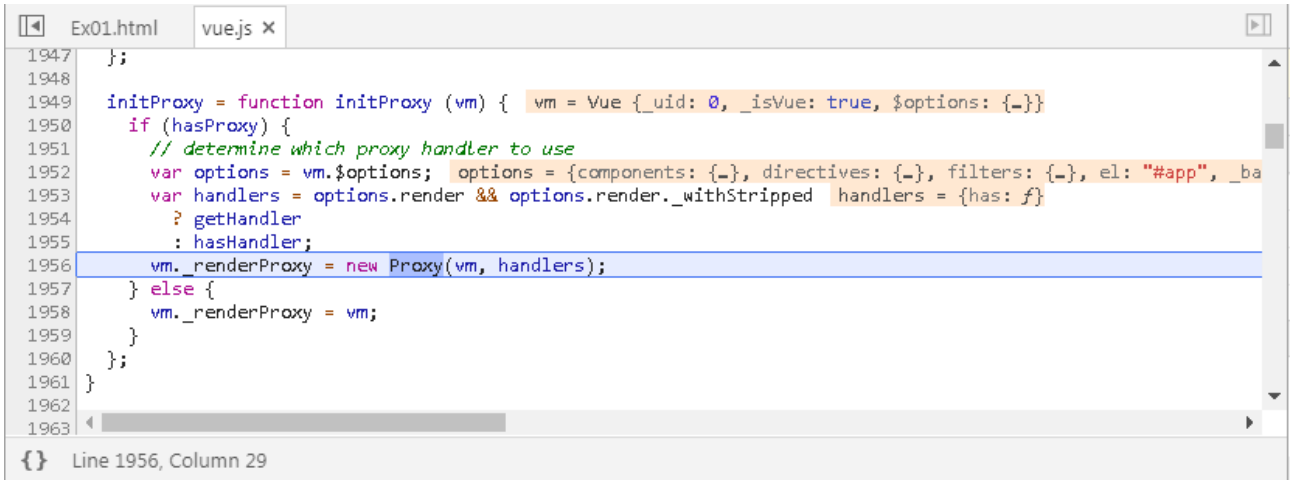
4. Algumas linhas abaixo é chamada a função initProxy, que controla o verdadeiro armazém de variáveis manipuladas pelo objeto Vue.



VUE – Substituição de variáveis

5. **Initproxy** é uma função que controla o ambiente de variáveis do Vue. Ela não é responsável pela observação constante das variáveis, função esta dos ***watchers***. Ela apenas promove o armazenamento e controle das variáveis em seções de interesse do Vue, como veremos. Normalmente, as variáveis globais do javascript pertencem ao objeto Window. O intuito do **proxy** de Vue é armazenar as variáveis no objeto Vue, e não no objeto Window. Ele também pretende substituir a função do objeto Proxy nativo dos navegadores Mozilla, pois outros navegadores não o possuem.

6. Instanciação do objeto Proxy. O objeto de apoio é **Vue**, na instância **vm**.

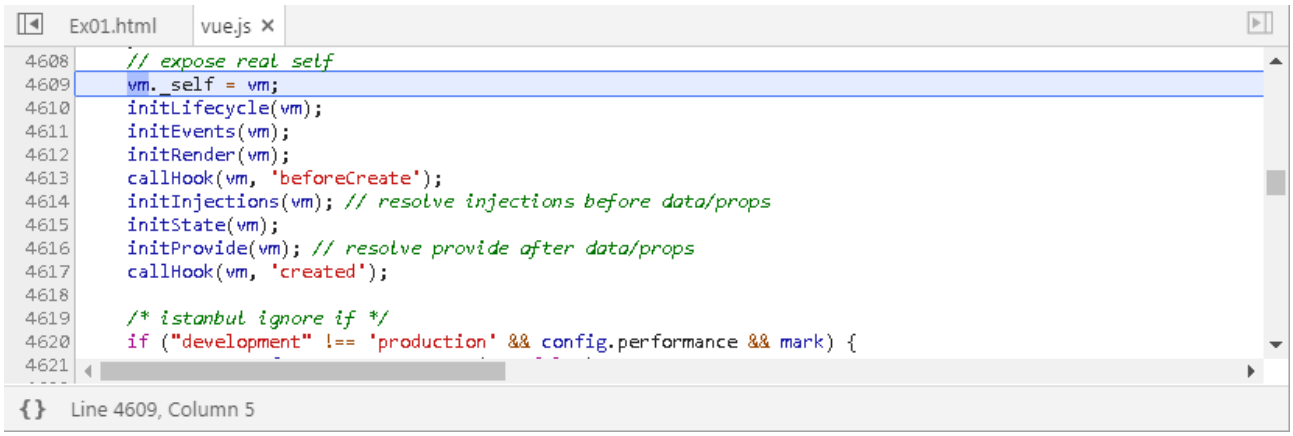


VUE – Substituição de variáveis

7. O que chamamos anteriormente de variáveis proxy é algo bem mais amplo. Além da seção data, temos components, directives, filters, etc, todos declarados nas opções de chamada ao inicializador do objeto Vue. Confira na linha $options deste objeto:

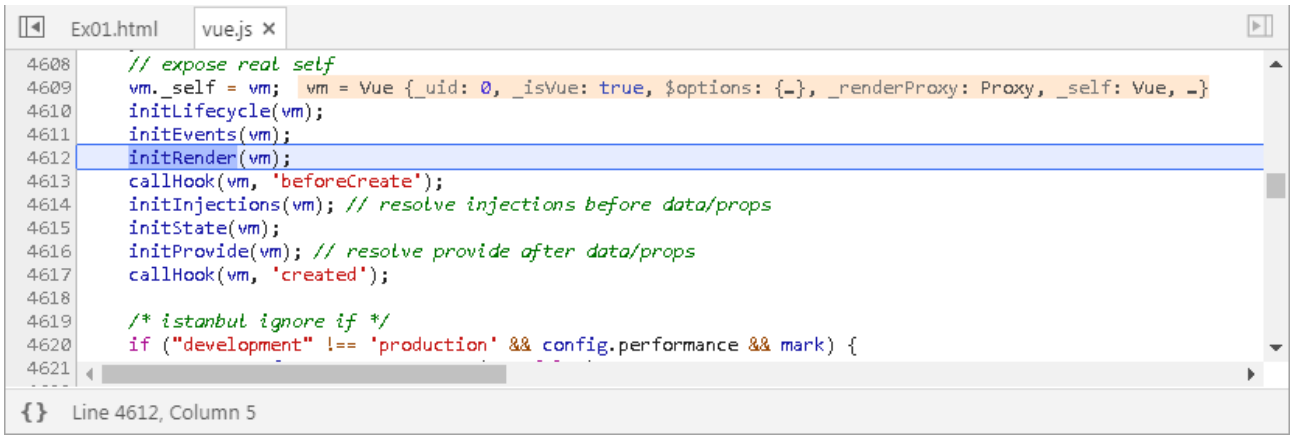
```
▼Proxy {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy}
  ▼[[Handler]]: Object
    ▶has: ƒ has(target, key)
    ▶__proto__: Object
  ▼[[Target]]: Vue
    ▶$options: {components: {…}, directives: {…}, filters: {…}, el: "#app", _base: ƒ, …}
      _isVue: true
    ▶_renderProxy: Proxy {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy}
      _uid: 0
      $data: (...)
      $isServer: (...)
      $props: (...)
      $ssrContext: (...)
    ▶__proto__: Object
   [[IsRevoked]]: false
```

8. Aqui está uma lista de funções de inicialização do Vue, que iniciam várias seções das instâncias de seus objetos, de acordo com as suas várias características e aspectos:



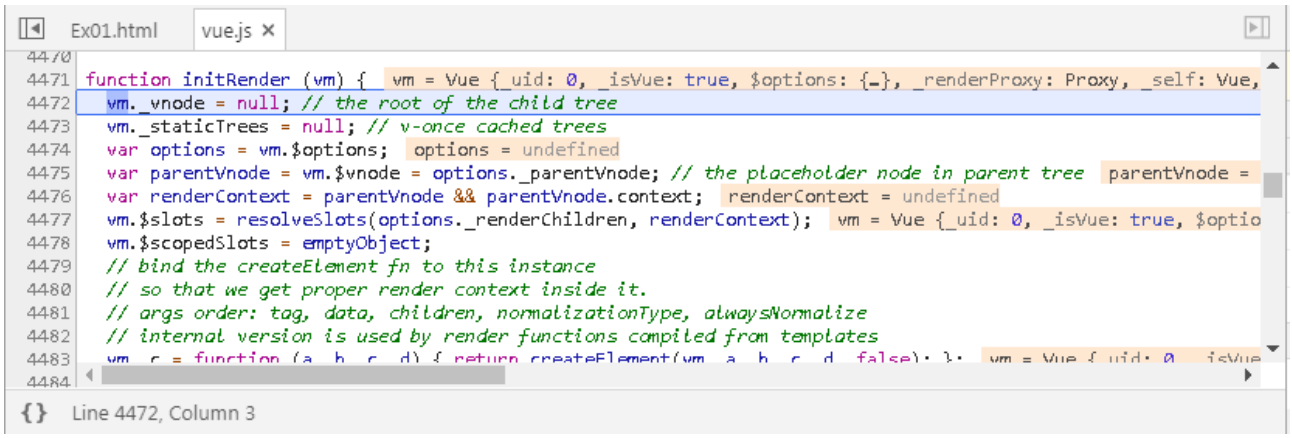
VUE – Substituição de variáveis

9. Vamos abordar apenas as inicializações mais importantes:



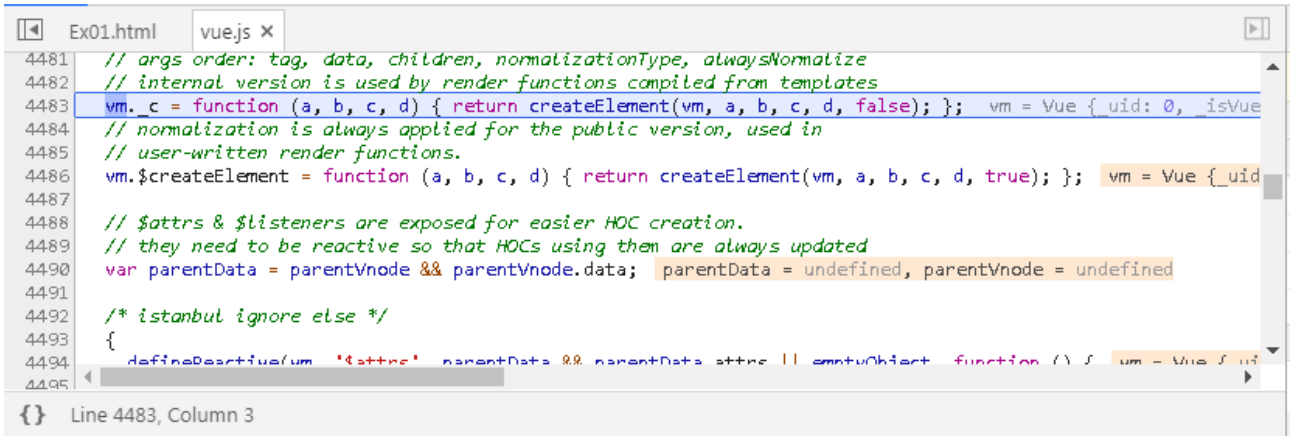
Iniciaremos com a função **initRender**.

10. A primeira parte faz algumas inicializações:

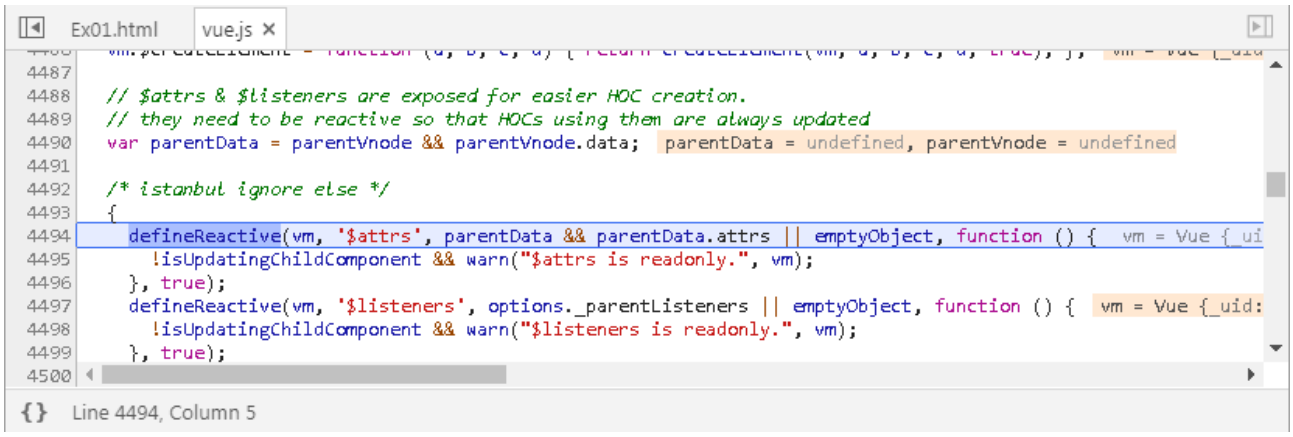


VUE – Substituição de variáveis

11. Uma das mais importantes definições se refere à função **vm._c**, principal função de renderização utilizada na apresentação de conteúdo no navegador, correspondente a ***createElement*** que, em última instância, chama a função de criação de um novo nodo HTML.

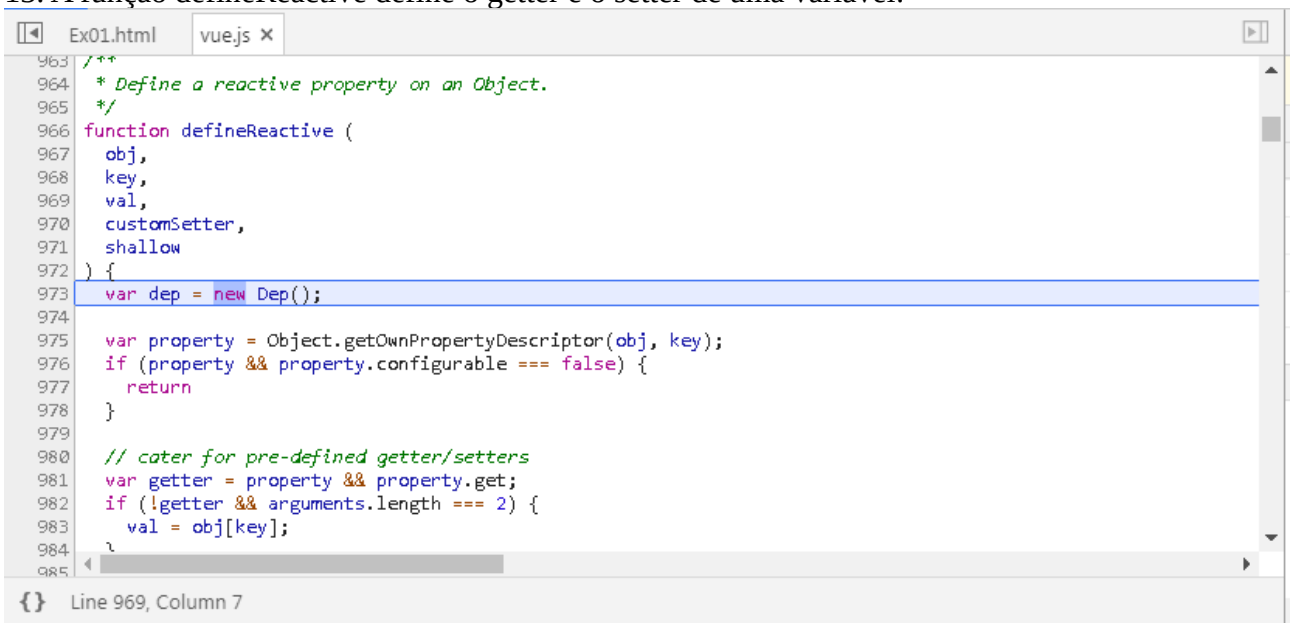


12. A função **initRender** também define duas variáveis básicas como “reativas”: $attrs e $listeners, fundamentais para funcionamento do VUE:



VUE – Substituição de variáveis

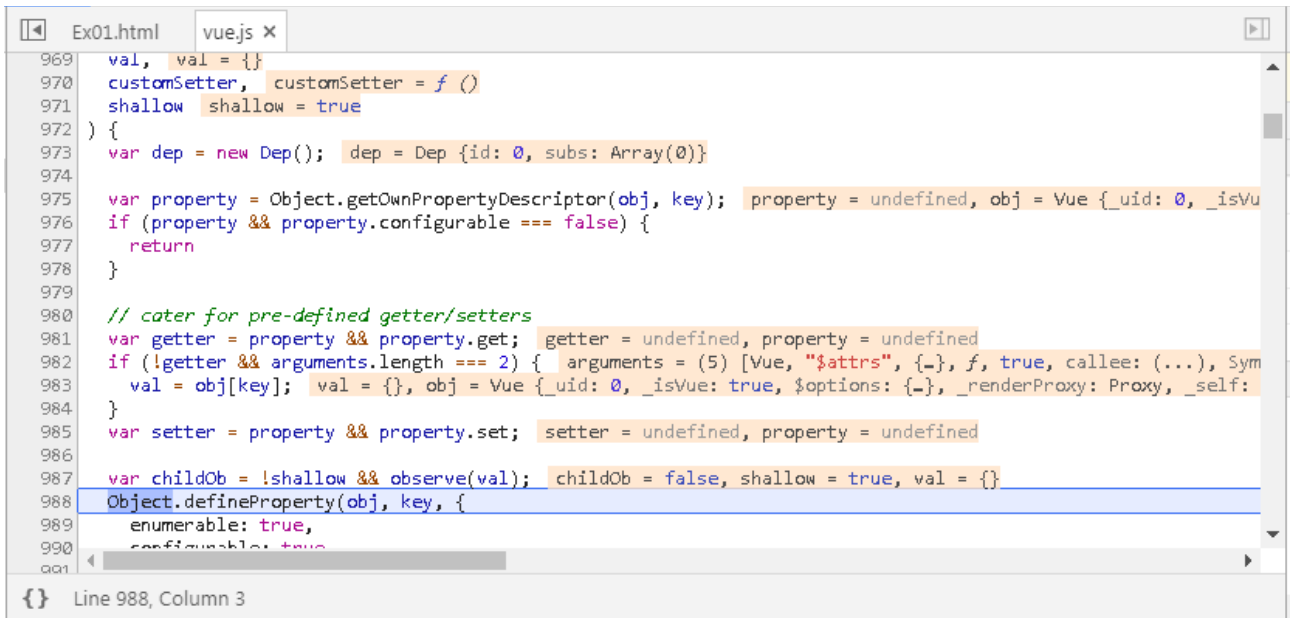
13. A função defineReactive define o getter e o setter de uma variável:



14. Em nosso caso ela acrescenta getter e setter de $attrs no objeto Vue, como mostrado abaixo:

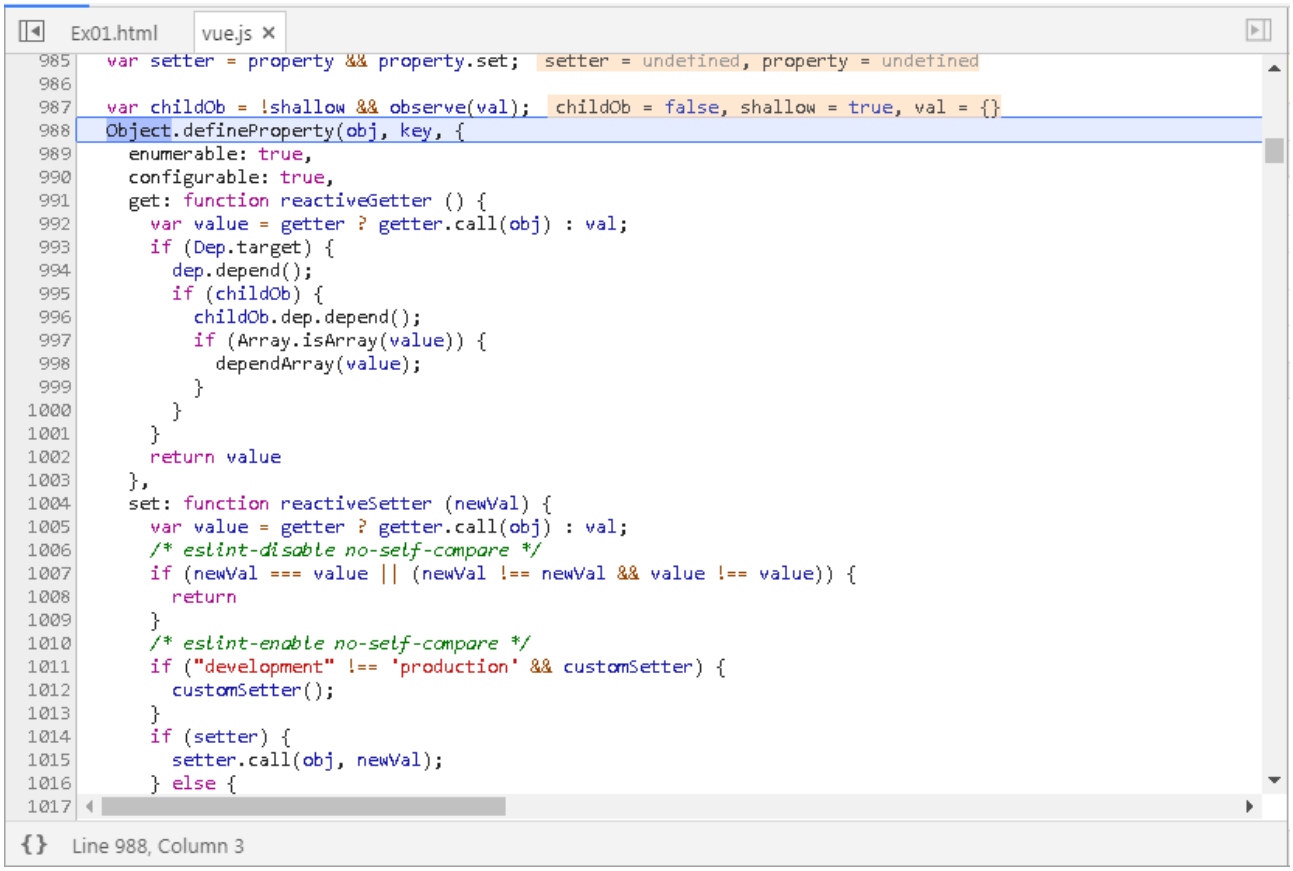
```
> obj
◀ ▶ Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
> key
"$attrs"
> val
◀ ▶ {}
```

15. Para isto é chamado o método **defineProperty** de Object:

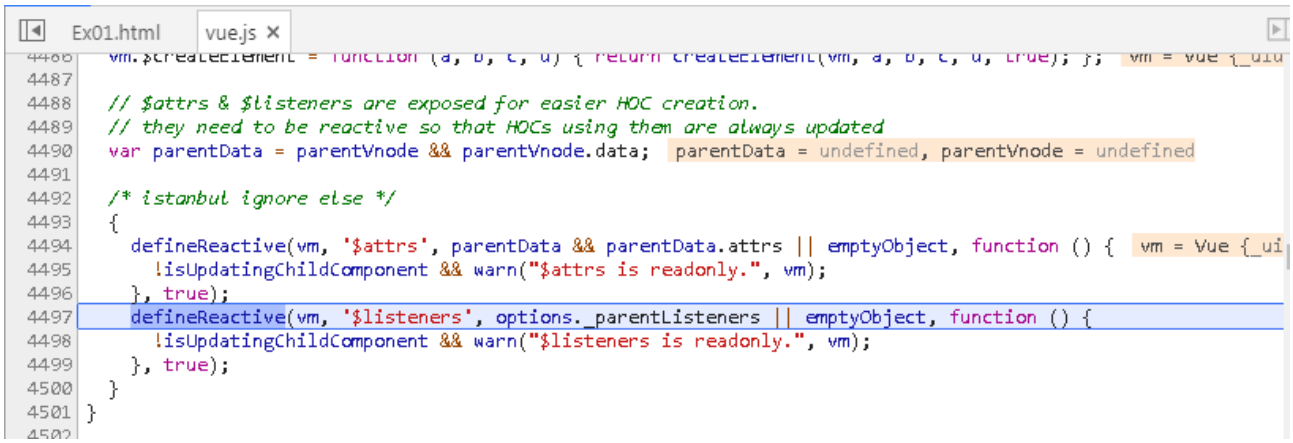


VUE – Substituição de variáveis

16. Nesta figura estão os métodos **get** e **set**, representados respectivamente por ***reactiveGetter*** e ***reactiveSetter***:

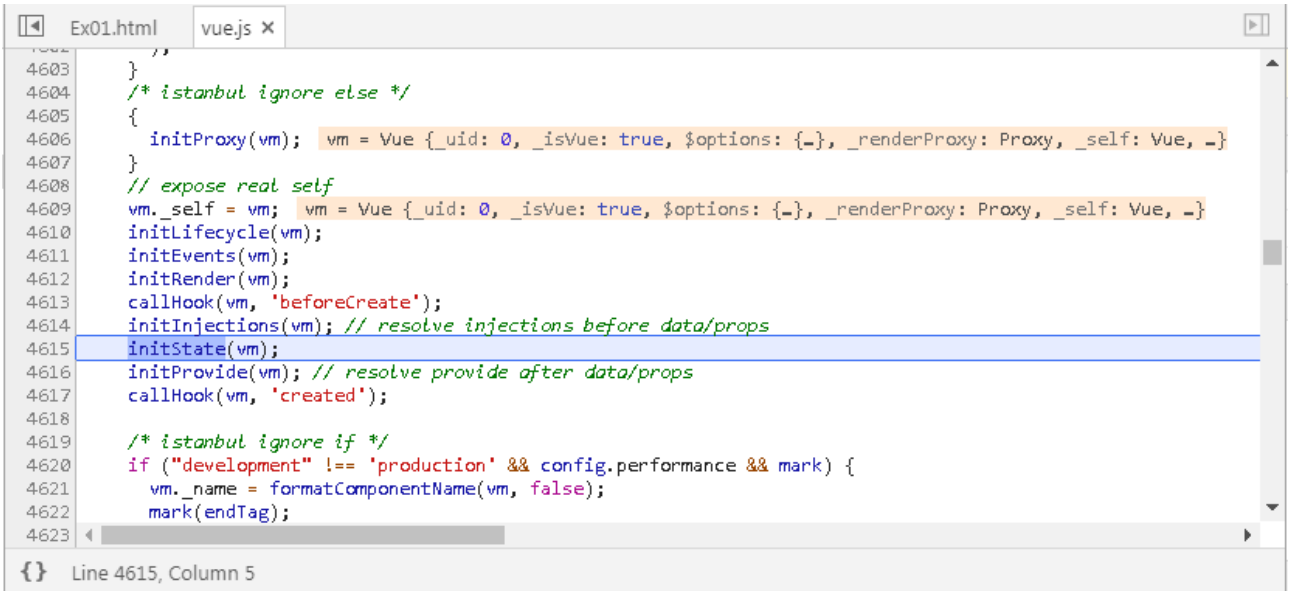


17. O mesmo é feito com a variável $listeners:



VUE – Substituição de variáveis

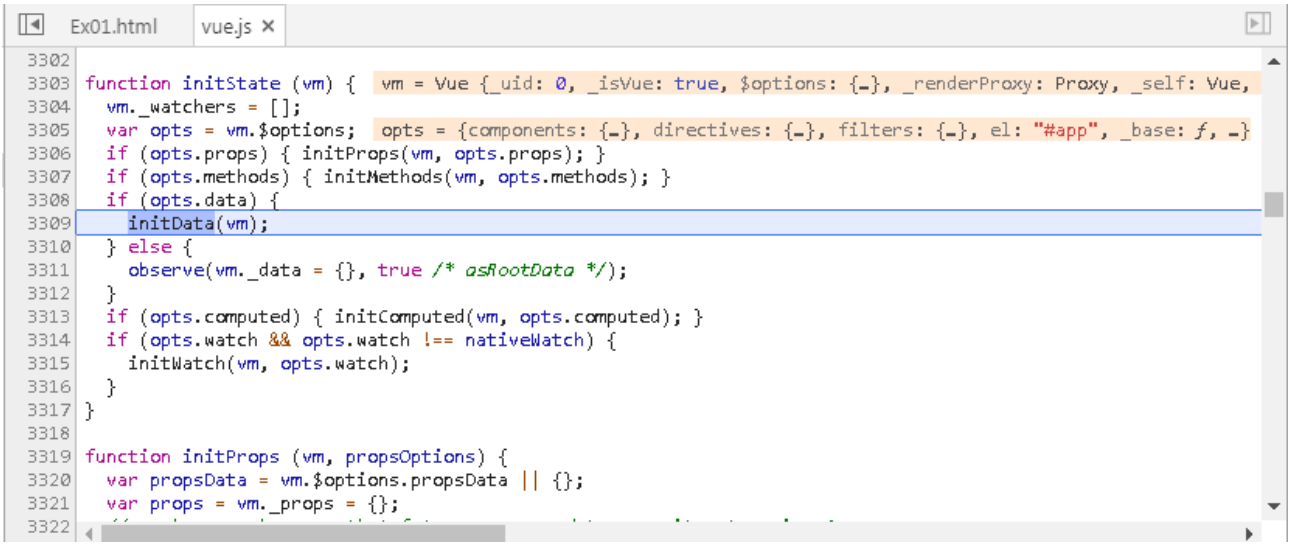
18. Vejamos agora a próxima função de inicialização importante é **initState**:



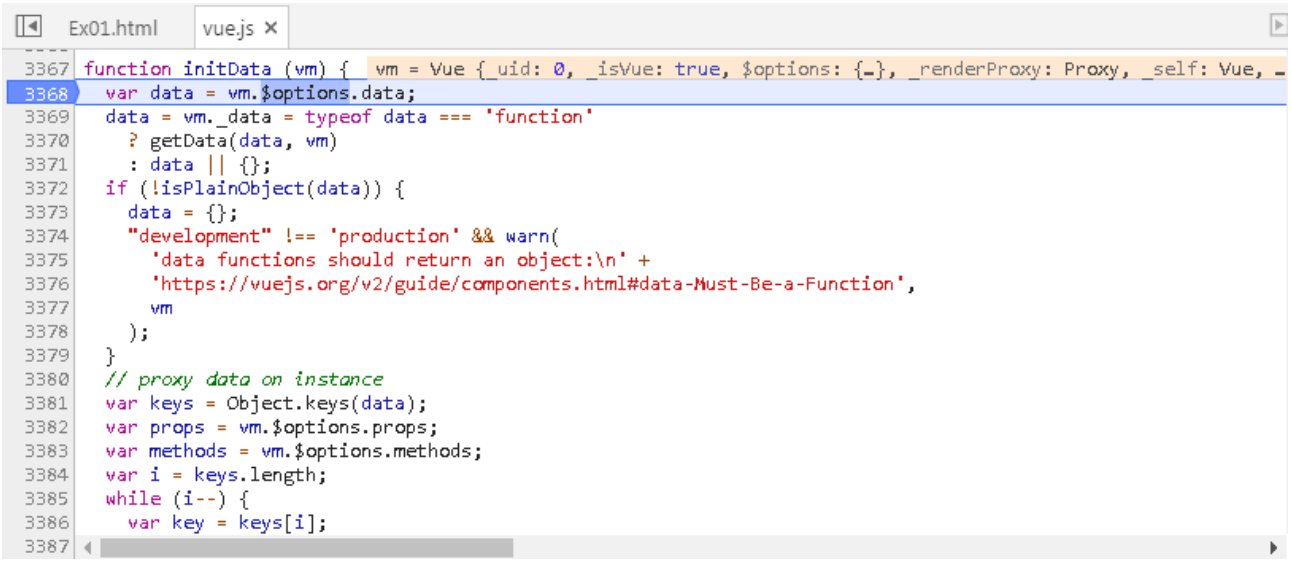
19. A função initState inicializa as variáveis (**data**), os **watchers**, as propriedades (**props**), métodos (**methods**) e as variáveis calculadas (**computed**):

VUE – Substituição de variáveis

20. Como o nosso exemplo só possui a seção data nas opções de chamada do Vue, o processamento salta para **initData**:



21. A primeira providência da função initData é coletar as opções declaradas na instanciação do Vue. Na linha seguinte (3369), a função **getData** é chamada, pelo fato de que a seção data é expressa internamente pela função **mergedInstanceDataFn**. Por isso, na figura posterior (22) o processamento prossegue em **getData**.



VUE – Substituição de variáveis

22. A função **getData** chama data por meio do método ***call***, para que não haja confusão com alguma variável data.

Ex01.html vue.js ×

```
3404    }
3405    // observe data
3406    observe(data, true /* asRootData */);
3407  }
3408
3409  function getData (data, vm) {  data = ƒ mergedInstanceDataFn(), vm = Vue {_uid: 0, _isVue: true, $options: {
3410    // #7573 disable dep collection when invoking data getters
3411    pushTarget();
3412    try {
3413      return data.call(vm, vm)
3414    } catch (e) {
3415      handleError(e, vm, "data()");
3416      return {}
3417    } finally {
3418      popTarget();
3419    }
3420  }
3421
3422  var computedWatcherOptions = { lazy: true };
3423
3424
```

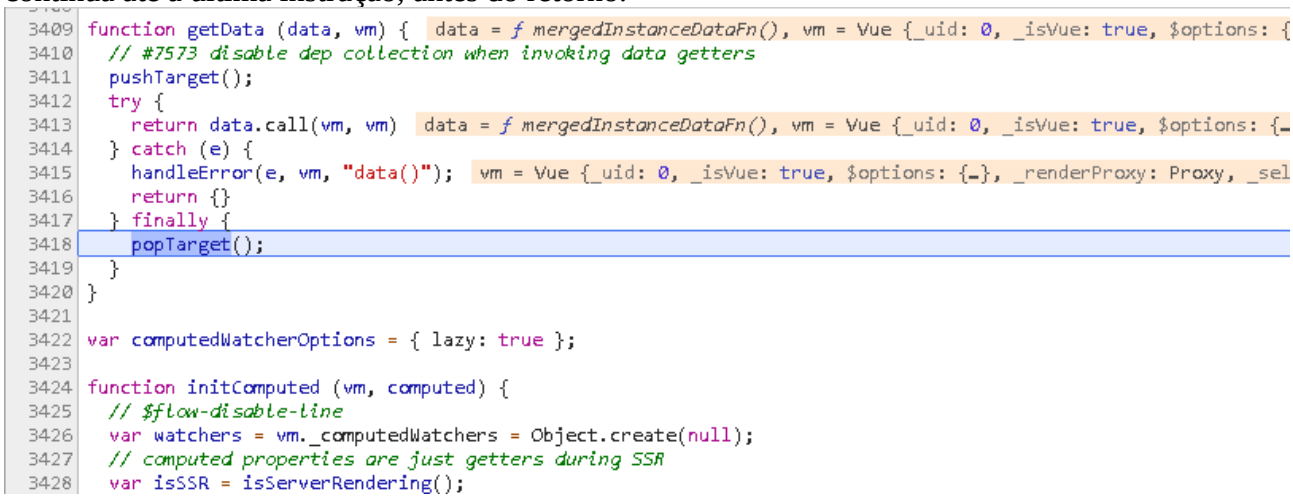
23. Em (**21**) esclarecemos que data havia sido substituído pela função mergedInstanceDataFn, daí esta ser a função chamada:

Ex01.html vue.js ×

```
1171      // it has to be a function to pass previous merges.
1172      return function mergedDataFn () {
1173        return mergeData(
1174          typeof childVal === 'function' ? childVal.call(this, this) : childVal,
1175          typeof parentVal === 'function' ? parentVal.call(this, this) : parentVal
1176        )
1177      }
1178    } else {
1179      return function mergedInstanceDataFn () {
1180        // instance merge
1181        var instanceData = typeof childVal === 'function'  instanceData = {author: "Meyer", message: "Hello Wo
1182          ? childVal.call(vm, vm)
1183          : childVal;
1184        var defaultData = typeof parentVal === 'function'  defaultData = undefined
1185          ? parentVal.call(vm, vm)
1186          : parentVal;
1187        if (instanceData) {  instanceData = {author: "Meyer", message: "Hello World!"}
1188          return mergeData(instanceData, defaultData)
1189        } else {
1190          return defaultData
```

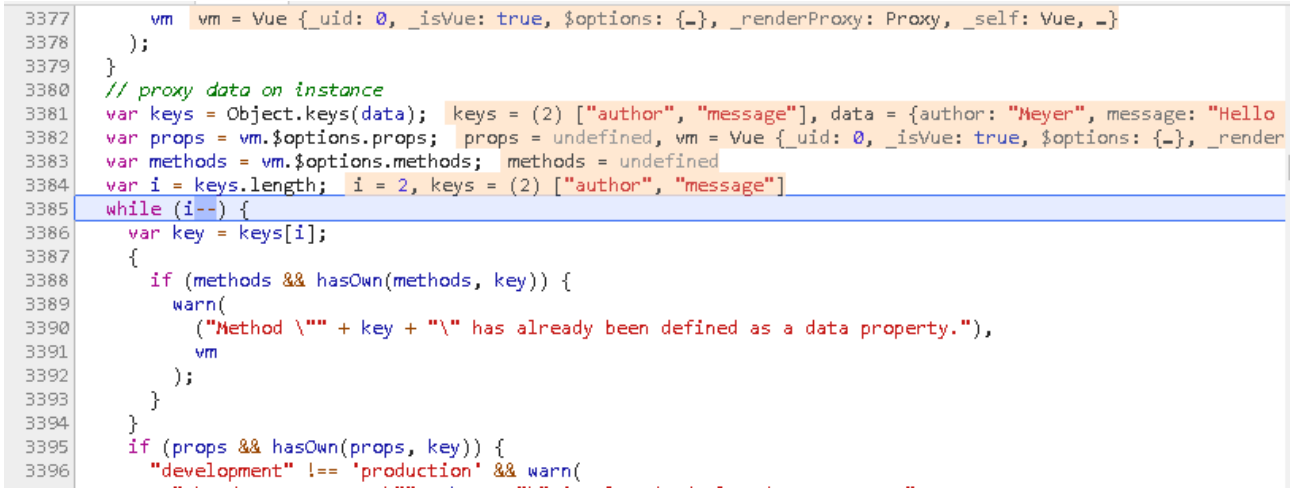
Esta funde a seção data default com a seção data fornecida na instanciação do Vue.

24. Após o processamento da função **mergedInstanceDataFn**, voltamos a **getData** e a função continua até a última instrução, antes do retorno.



```
3409 function getData (data, vm) {   data = ƒ mergedInstanceDataFn(), vm = Vue {_uid: 0, _isVue: true, $options: {
3410   // #7573 disable dep collection when invoking data getters
3411   pushTarget();
3412   try {
3413     return data.call(vm, vm)   data = ƒ mergedInstanceDataFn(), vm = Vue {_uid: 0, _isVue: true, $options: {…
3414   } catch (e) {
3415     handleError(e, vm, "data()");   vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _sel
3416     return {}
3417   } finally {
3418     popTarget();
3419   }
3420 }
3421
3422 var computedWatcherOptions = { lazy: true };
3423
3424 function initComputed (vm, computed) {
3425   // $flow-disable-line
3426   var watchers = vm._computedWatchers = Object.create(null);
3427   // computed properties are just getters during SSR
3428   var isSSR = isServerRendering();
```

25. O processamento retorna para a função initData, onde outras providências são tomadas, e entre elas a que vai nos interessar, no momento, é o processamento das chaves que armazenam os conteúdos da seção data (“***author***” e “***message***”):



```
3377       vm   vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
3378     );
3379   }
3380   // proxy data on instance
3381   var keys = Object.keys(data);   keys = (2) ["author", "message"], data = {author: "Meyer", message: "Hello
3382   var props = vm.$options.props;   props = undefined, vm = Vue {_uid: 0, _isVue: true, $options: {…}, _render
3383   var methods = vm.$options.methods;   methods = undefined
3384   var i = keys.length;   i = 2, keys = (2) ["author", "message"]
3385   while (i--) {
3386     var key = keys[i];
3387     {
3388       if (methods && hasOwn(methods, key)) {
3389         warn(
3390           ("Method \"" + key + "\" has already been defined as a data property."),
3391           vm
3392         );
3393       }
3394     }
3395     if (props && hasOwn(props, key)) {
3396       "development" !== 'production' && warn(
```

26. E aqui chegamos ao centro do nosso objetivo: a chamada à função **proxy**:

```
3386    var key = keys[i];  key = "message", keys = (2) ["author", "message"]
3387    {
3388      if (methods && hasOwn(methods, key)) {  methods = undefined, key = "message"
3389        warn(
3390          ("Method \"" + key + "\" has already been defined as a data property."),  key = "message"
3391          vm  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
3392        );
3393      }
3394    }
3395    if (props && hasOwn(props, key)) {  props = undefined, key = "message"
3396      "development" !== 'production' && warn(
3397        "The data property \"" + key + "\" is already declared as a prop. " +  key = "message"
3398        "Use prop default value instead.",
3399        vm  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
3400      );
3401    } else if (!isReserved(key)) {  key = "message"
3402      proxy(vm, "_data", key);
3403    }
3404  }
3405  // observe data
```

27. A função **proxy** define **set** e **get** para cada variável que lhe é apresentada, com base, respectivamente, em duas funções: ***proxySetter*** e ***proxyGetter***. As variáveis ficarão na seção "_data" do objeto Vue.

```
3289    get: noop,
3290    set: noop
3291  };
3292
3293  function proxy (target, sourceKey, key) {  target = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy:
3294    sharedPropertyDefinition.get = function proxyGetter () {
3295      return this[sourceKey][key]  sourceKey = "_data", key = "message"
3296    };
3297    sharedPropertyDefinition.set = function proxySetter (val) {
3298      this[sourceKey][key] = val;  sourceKey = "_data", key = "message"
3299    };
3300    Object.defineProperty(target, key, sharedPropertyDefinition);  target = Vue {_uid: 0, _isVue: true, $optio
3301  }
3302
```

28. Ao final, a função ***proxy*** retorna a **initData**. Em seguida é chamada a função **observe**:

```
3397          "The data property \"" + key + "\" is already declared as a prop. " +  key = "author"
3398          "Use prop default value instead.",
3399          vm  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
3400        );
3401      } else if (!isReserved(key)) {  key = "author"
3402        proxy(vm, "_data", key);  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self:
3403      }
3404    }
3405    // observe data
3406    observe(data, true /* asRootData */);
3407  }
3408
3409  function getData (data, vm) {
3410    // #7573 disable dep collection when invoking data getters
3411    pushTarget();
3412    try {
3413      return data.call(vm, vm)
3414    } catch (e) {
3415      handleError(e, vm, "data()");
3416      return {}
```

VUE – Substituição de variáveis

29. A função observe vai criar uma nova instância do objeto **Observer** para as variáveis da seção data do objeto instanciado Vue:

```
940   */
941  function observe (value, asRootData) {  value = {author: "Meyer", message: "Hello World!"}, asRootData = tru
942    if (!isObject(value) || value instanceof VNode) {
943      return
944    }
945    var ob;  ob = undefined
946    if (hasOwn(value, '__ob__') && value.__ob__ instanceof Observer) {  value = {author: "Meyer", message: "He
947      ob = value.__ob__;  ob = undefined
948    } else if (
949      shouldObserve &&
950      !isServerRendering() &&
951      (Array.isArray(value) || isPlainObject(value)) &&  value = {author: "Meyer", message: "Hello World!"}
952      Object.isExtensible(value) &&
953      !value._isVue
954    ) {
955      ob = new Observer(value);
956    }
957    if (asRootData && ob) {
958      ob.vmCount++;
959    }
```

30. A função Observer cria a definição de um objeto Observer que recebe as variáveis Vue e as insere em uma definição, como mostrado:

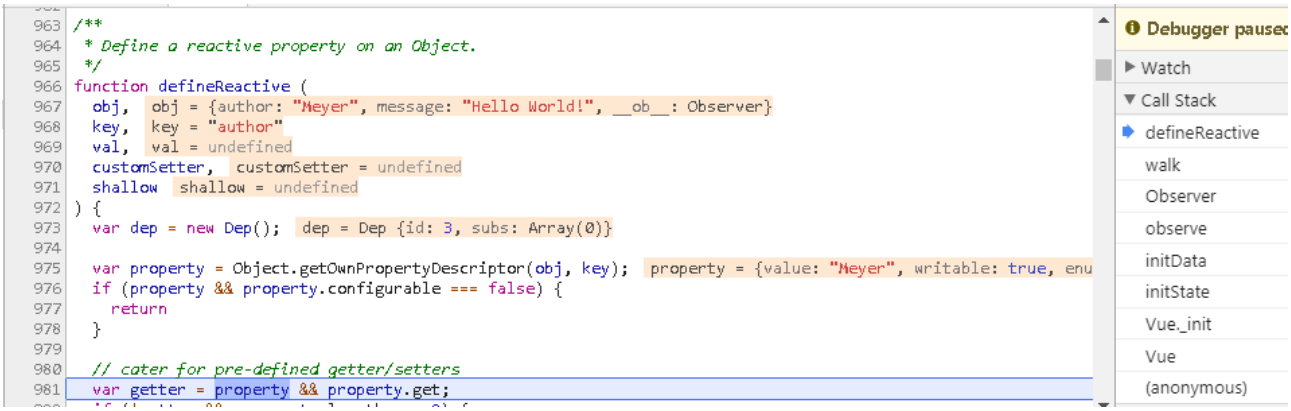
```
871   * object. Once attached, the observer converts the target
872   * object's property keys into getter/setters that
873   * collect dependencies and dispatch updates.
874   */
875  var Observer = function Observer (value) {  value = {author: "Meyer", message: "Hello World!", __ob__: Obser
876    this.value = value;
877    this.dep = new Dep();
878    this.vmCount = 0;
879    def(value, '__ob__', this);  value = {author: "Meyer", message: "Hello World!", __ob__: Observer}
880    if (Array.isArray(value)) {
881      var augment = hasProto  augment = undefined
882        ? protoAugment
883        : copyAugment;
884      augment(value, arrayMethods, arrayKeys);  augment = undefined, value = {author: "Meyer", message: "Hello
885      this.observeArray(value);
886    } else {
887      this.walk(value);  value = {author: "Meyer", message: "Hello World!", __ob__: Observer}
888    }
889  };
```

Depois é chamada a função walk desta definição de objeto.

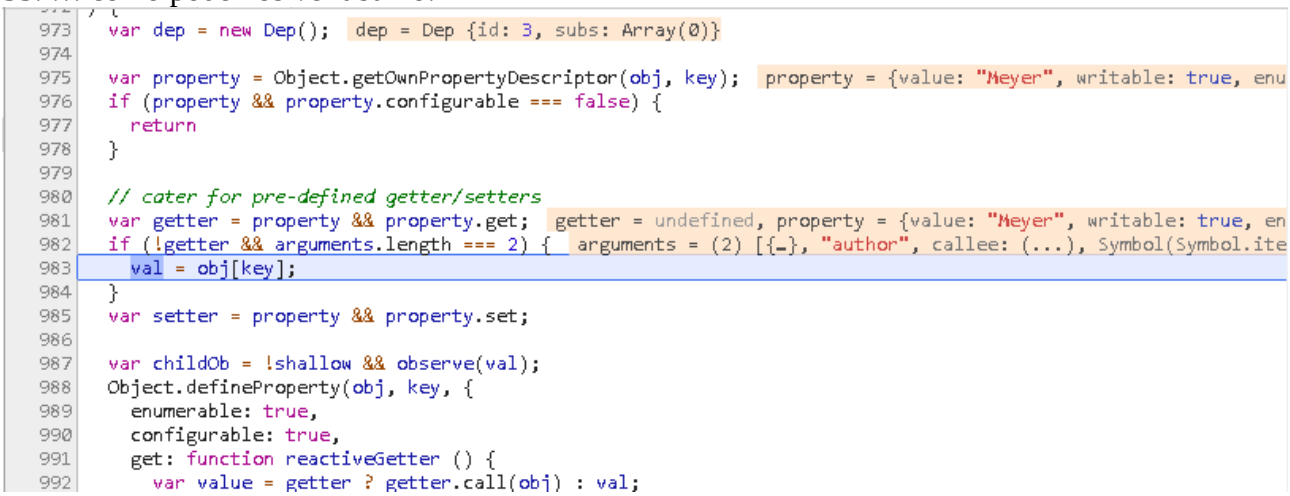
31. Desta vez, não é apenas um objeto **Proxy** que é definido como "reativo", e sim um objeto **Observer**, que descreve um conjunto de variáveis Vue:

VUE – Substituição de variáveis

32. São definidos os ***getters*** e ***setters*** para este objeto **Observer**:



33. … como podemos ver abaixo:



A linha 987 provoca a chamada da função **observe**.

VUE – Substituição de variáveis

34. Inspecionando o objeto Observer:

```
> obj
< ▶ {author: "Meyer", message: "Hello World!", __ob__: Observer}
> key
< "author"
> obj[key]
< "Meyer"
```

35. A função ***observe*** vai definir uma instância do objeto **Observer**:

```
937  * Attempt to create an observer instance for a value,
938  * returns the new observer if successfully observed,
939  * or the existing observer if the value already has one.
940  */
941 function observe (value, asRootData) {  value = "Meyer", asRootData = undefined
942   if (!isObject(value) || value instanceof VNode) {
943     return
944   }
945   var ob;  ob = undefined
946   if (hasOwn(value, '__ob__') && value.__ob__ instanceof Observer) {  value = "Meyer"
947     ob = value.__ob__;  ob = undefined
948   } else if (
949     shouldObserve &&
950     !isServerRendering() &&
951     (Array.isArray(value) || isPlainObject(value)) &&  value = "Meyer"
952     Object.isExtensible(value) &&
953     !value._isVue
954   ) {
955     ob = new Observer(value);  ob = undefined, value = "Meyer"
```

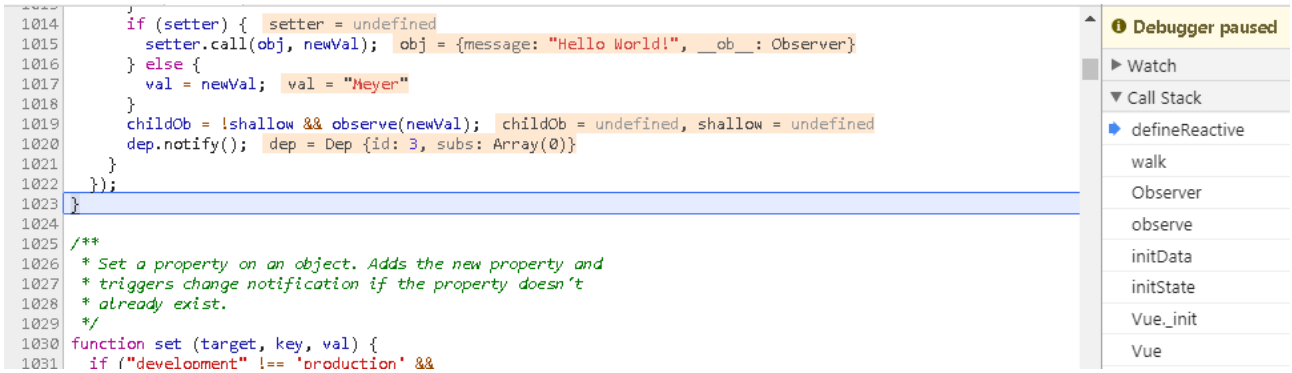
36. Como io valor de *author* não é um objeto, não é instanciado um objeto **Observer** para ele. O mesmo acontecerá com *message*:

Ex01.html vue.js ×

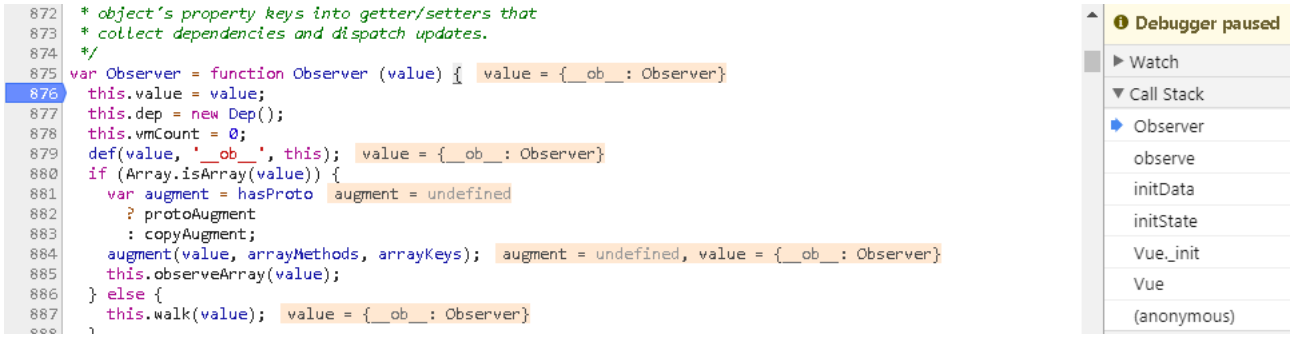
```
937  * Attempt to create an observer instance for a value,
938  * returns the new observer if successfully observed,
939  * or the existing observer if the value already has one.
940  */
941 function observe (value, asRootData) {  value = "Meyer", asRootData = undefined
942   if (!isObject(value) || value instanceof VNode) {
943     return
944   }
```

VUE – Substituição de variáveis

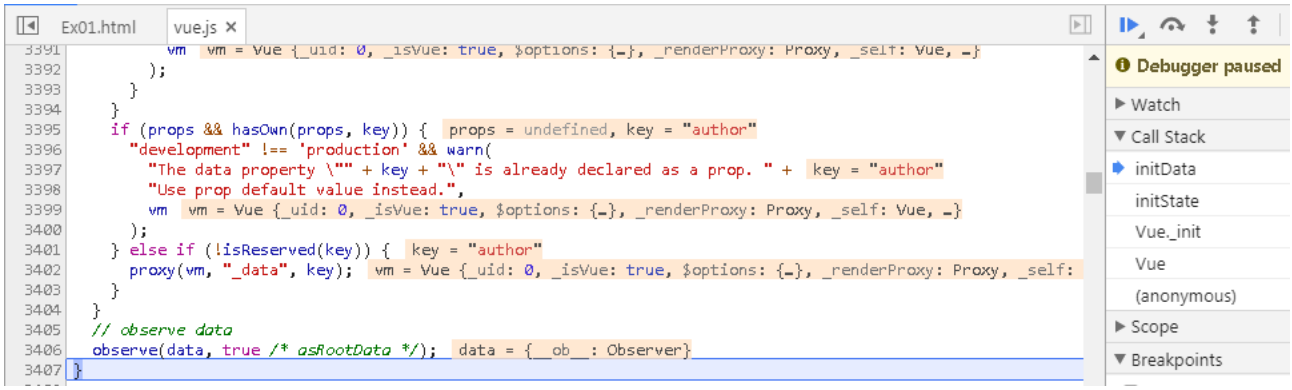
37. Chegamos ao fim da função ***defineReactive***.



38. Retornamos à função Observer:

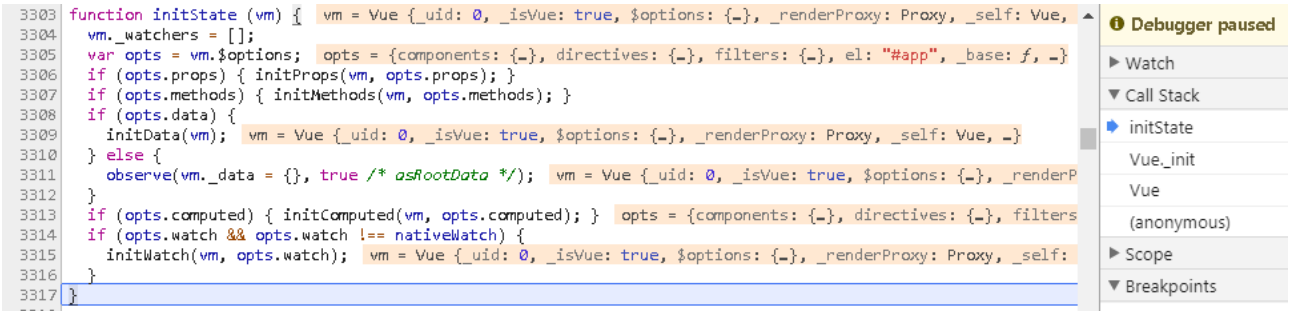


39. Retornamos à função ***initData***:

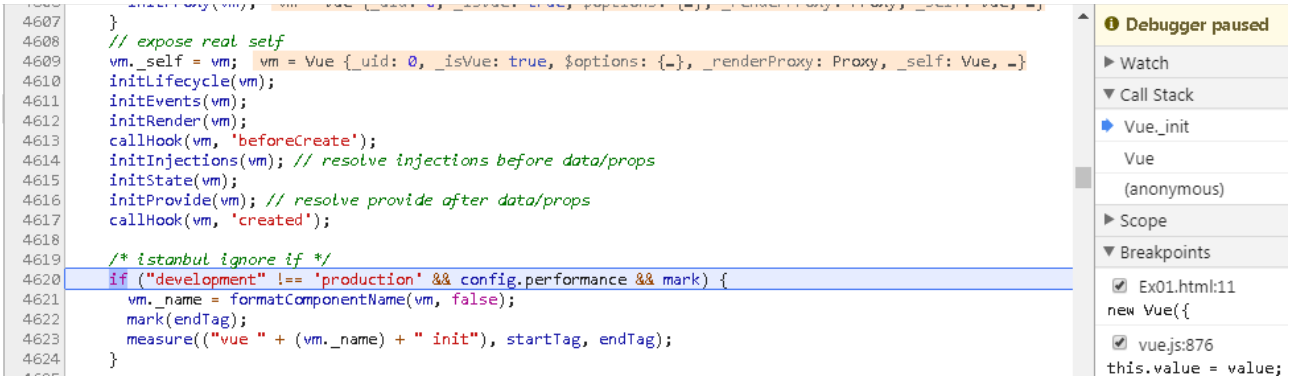


VUE – Substituição de variáveis

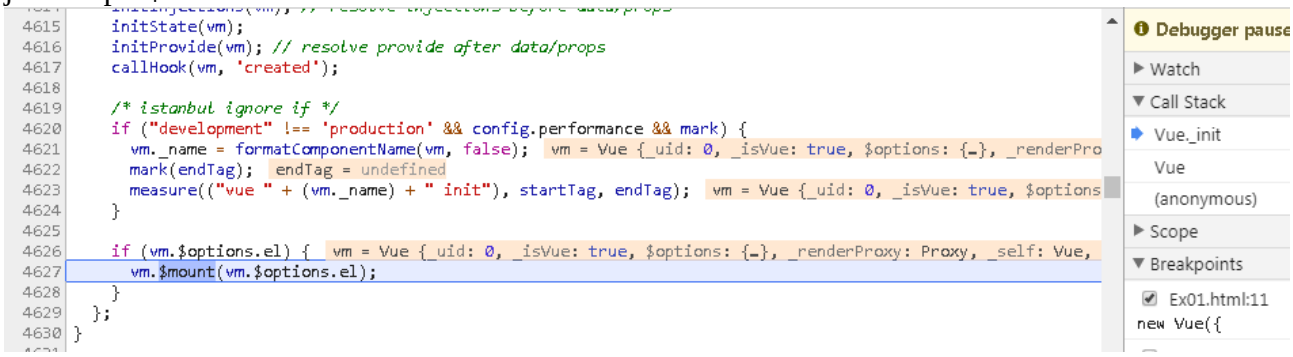
40. Retornamos à função ***initState***:



41. De volta à **Vue._init**, outras duas funções são executadas, sem importância para o momento (initProvide e Callhook para o evento 'created').

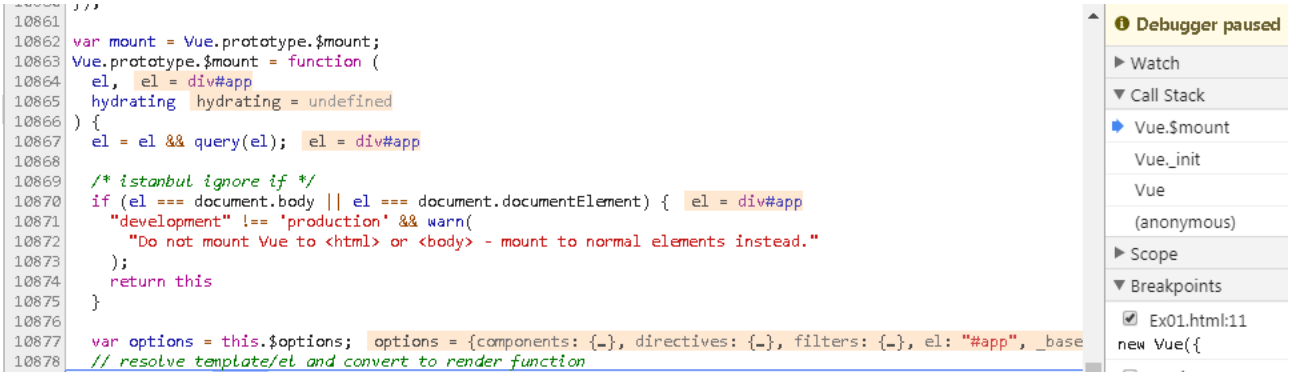


42. Na linha 4627 é chamada a função identidade do Vue por excelência, responsável pelo reconhecimento dos contextos no código HTML que convive com a chamada corrente de Vue no javascript: **$mount**.



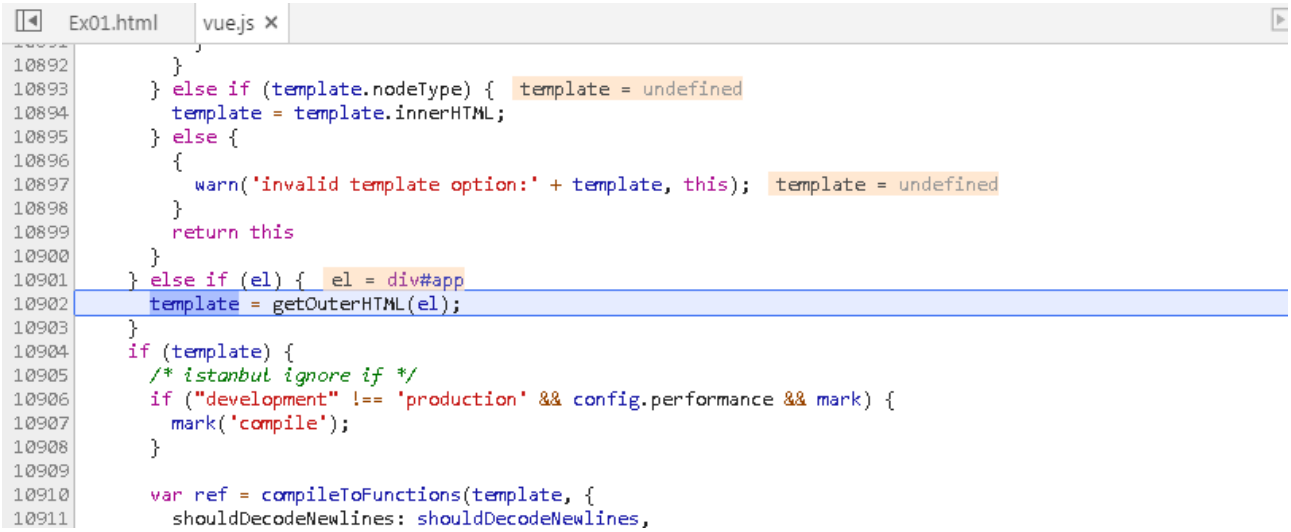
VUE – Substituição de variáveis

43. Nos parâmetros da chamada (linha 10864) se vê claramente a identificação do contexto de "montagem" da instância do objeto Vue: ***"div#app"***.



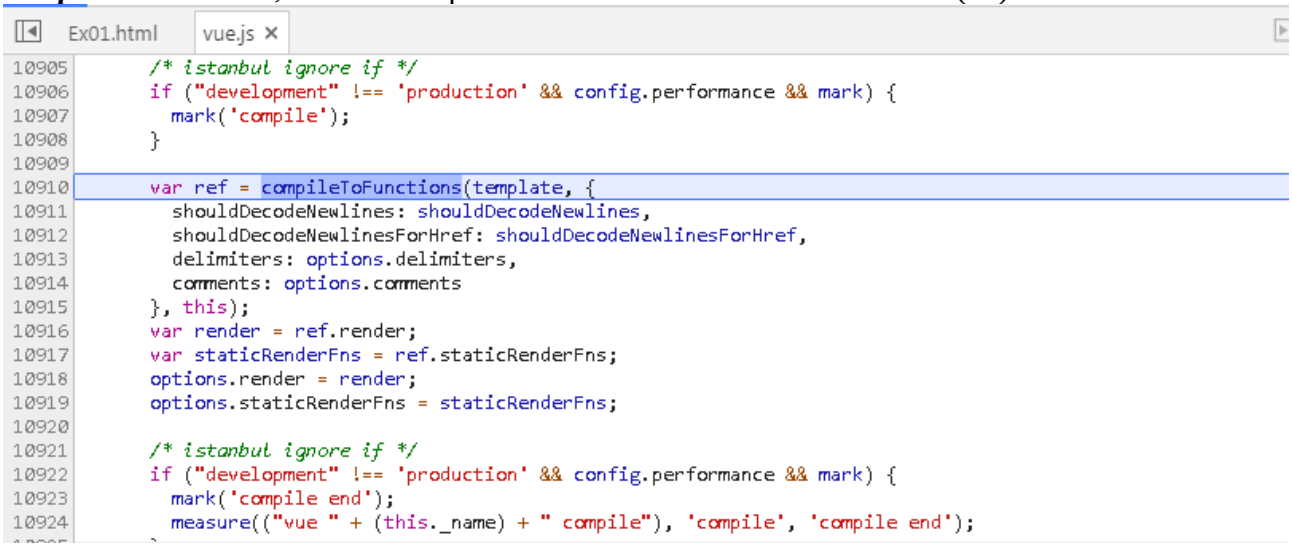

```
10861
10862 var mount = Vue.prototype.$mount;
10863 Vue.prototype.$mount = function (
10864   el,  el = div#app
10865   hydrating  hydrating = undefined
10866 ) {
10867   el = el && query(el);  el = div#app
10868
10869   /* istanbul ignore if */
10870   if (el === document.body || el === document.documentElement) {  el = div#app
10871     "development" !== 'production' && warn(
10872       "Do not mount Vue to <html> or <body> - mount to normal elements instead."
10873     );
10874     return this
10875   }
10876
10877   var options = this.$options;  options = {components: {…}, directives: {…}, filters: {…}, el: "#app", _base
10878   // resolve template/el and convert to render function
```

44. Na linha **10902**, na falta de uma tag template, ou de um contexto de componente, o corpo do HTML contido no contexto identificado na chamada do Vue é estabelecido como a template para o Vue, através da chamada a ***getOuterHTML***:




```
10892         }
10893       } else if (template.nodeType) {  template = undefined
10894         template = template.innerHTML;
10895       } else {
10896         {
10897           warn('invalid template option:' + template, this);  template = undefined
10898         }
10899         return this
10900       }
10901     } else if (el) {  el = div#app
10902       template = getOuterHTML(el);
10903     }
10904     if (template) {
10905       /* istanbul ignore if */
10906       if ("development" !== 'production' && config.performance && mark) {
10907         mark('compile');
10908       }
10909
10910       var ref = compileToFunctions(template, {
10911         shouldDecodeNewlines: shouldDecodeNewlines,
```

45. Para destrinchar a template, seja explícita ou apenas de contexto (**"el"**), o Vue chama a função ***compileToFunctions***, tendo como parâmetro o trecho HTML extraído em (44):




```
10905       /* istanbul ignore if */
10906       if ("development" !== 'production' && config.performance && mark) {
10907         mark('compile');
10908       }
10909
10910       var ref = compileToFunctions(template, {
10911         shouldDecodeNewlines: shouldDecodeNewlines,
10912         shouldDecodeNewlinesForHref: shouldDecodeNewlinesForHref,
10913         delimiters: options.delimiters,
10914         comments: options.comments
10915       }, this);
10916       var render = ref.render;
10917       var staticRenderFns = ref.staticRenderFns;
10918       options.render = render;
10919       options.staticRenderFns = staticRenderFns;
10920
10921       /* istanbul ignore if */
10922       if ("development" !== 'production' && config.performance && mark) {
10923         mark('compile end');
10924         measure(("vue " + (this._name) + " compile"), 'compile', 'compile end');
```

VUE – Substituição de variáveis

46. A função coleta as opções de compilação do código HTML do contexto Vue:

```
10674 }
10675
10676 function createCompileToFunctionFn (compile) {
10677   var cache = Object.create(null);
10678
10679   return function compileToFunctions (
10680     template,  template = "<div id="app">{{ author }} says: {{ message }}</div>"
10681     options,  options = {shouldDecodeNewlines: false, shouldDecodeNewlinesForHref: true, delimiters: undefin
10682     vm  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
10683   ) {
10684     options = extend({}, options);  options = {shouldDecodeNewlines: false, shouldDecodeNewlinesForHref: tru
10685     var warn$$1 = options.warn || warn;  warn$$1 = undefined
10686     delete options.warn;
10687
10688     /* istanbul ignore if */
10689     {
10690       // detect possible CSP restriction
10691       try {
10692         new Function('return 1');
10693       } catch (e) {
```

47. E, por sua vez, esta função chama a função compile, tendo como parâmetro também o código HTML do contexto Vue almejado:

```
10704     }
10705
10706     // check cache
10707     var key = options.delimiters  key = "<div id="app">{{ author }} says: {{ message }}</div>", options = {s
10708       ? String(options.delimiters) + template  template = "<div id="app">{{ author }} says: {{ message }}</d
10709       : template;
10710     if (cache[key]) {  key = "<div id="app">{{ author }} says: {{ message }}</div>"
10711       return cache[key]
10712     }
10713
10714     // compile
10715     var compiled = compile(template, options);  compiled = undefined, template = "<div id="app">{{ author }}
10716
10717     // check compilation errors/tips
10718     {
10719       if (compiled.errors && compiled.errors.length) {  compiled = undefined
10720         warn$$1(  warn$$1 = ƒ (msg, vm)
10721           "Error compiling template:\n\n" + template + "\n\n" +  template = "<div id="app">{{ author }} says
10722           compiled.errors.map(function (e) { return ("- " + e); }).join('\n') + '\n',  compiled = undefined
10723           vm  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
```

48. A função ***compile*** fica em um contexto comum às chamadas de ***createCompiler***, inserido, por sua vez, no contexto de chamada de ***createCompilerCreator***:

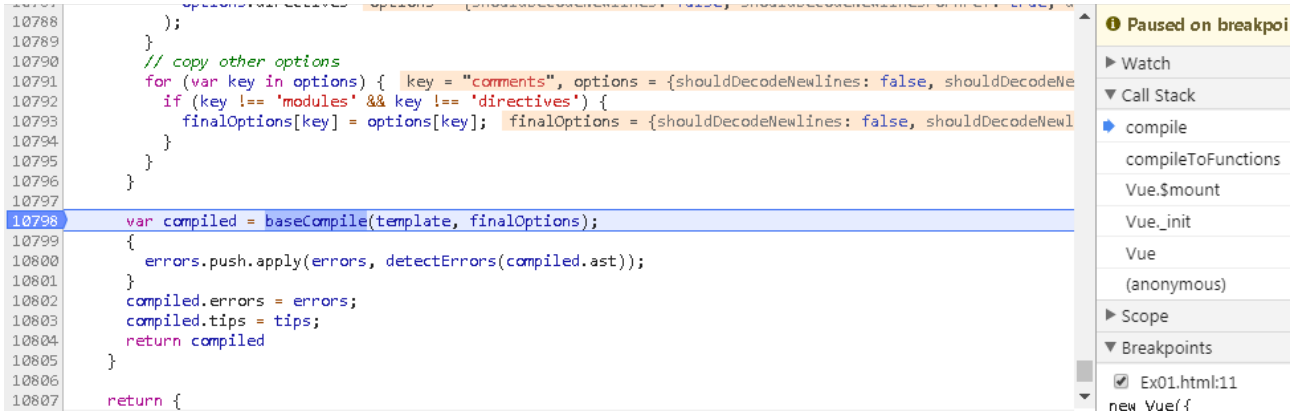
```
/*  */

function createCompilerCreator (baseCompile) {
  return function createCompiler (baseOptions) {
    function compile (
      template,  template = "<div id="app">{{ author }} says: {{ message }}</div>"
      options  options = {shouldDecodeNewlines: false, shouldDecodeNewlinesForHref: true, delimiters: undefi
    ) {
      var finalOptions = Object.create(baseOptions);  finalOptions = {warn: ƒ}
      var errors = [];  errors = []
      var tips = [];  tips = []
      finalOptions.warn = function (msg, tip) {  finalOptions = {warn: ƒ}
        (tip ? tips : errors).push(msg);  tips = [], errors = []
      };

      if (options) {  options = {shouldDecodeNewlines: false, shouldDecodeNewlinesForHref: true, delimiters:
        // merge custom modules
        if (options.modules) {  options = {shouldDecodeNewlines: false, shouldDecodeNewlinesForHref: true, d
          finalOptions.modules =  finalOptions = {warn: ƒ}
            (baseOptions.modules || []).concat(options.modules);  options = {shouldDecodeNewlines: false, sh
```

Debugger paused
▶ Watch
▼ Call Stack
compile
compileToFunctions
Vue.$mount
Vue._init
Vue
(anonymous)
▶ Scope
▼ Breakpoints
Ex01.html:11

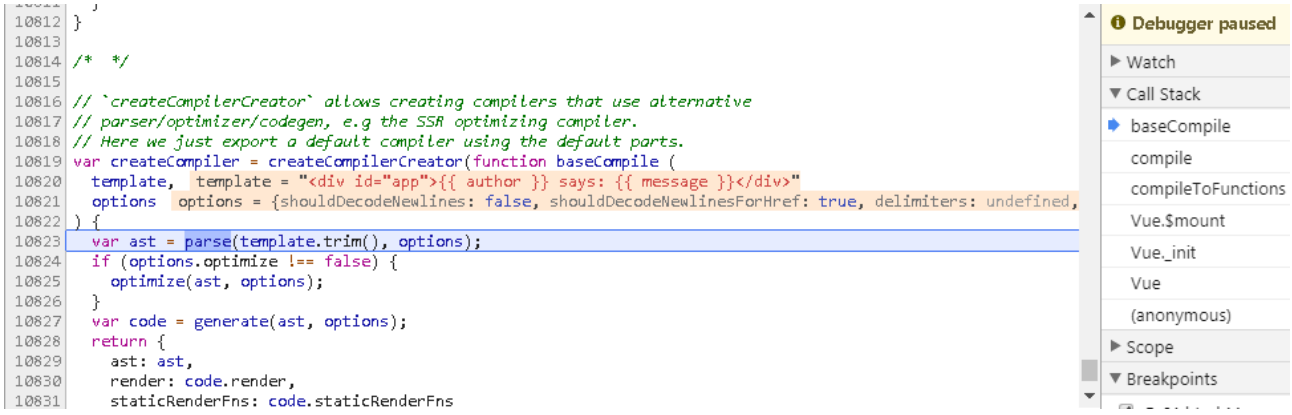
VUE – Substituição de variáveis

49. A função ***compile*** checa quais seções de opções foram fornecidas na instanciação do Vue:

50. E invoca ***baseCompile***, com a mesma template:



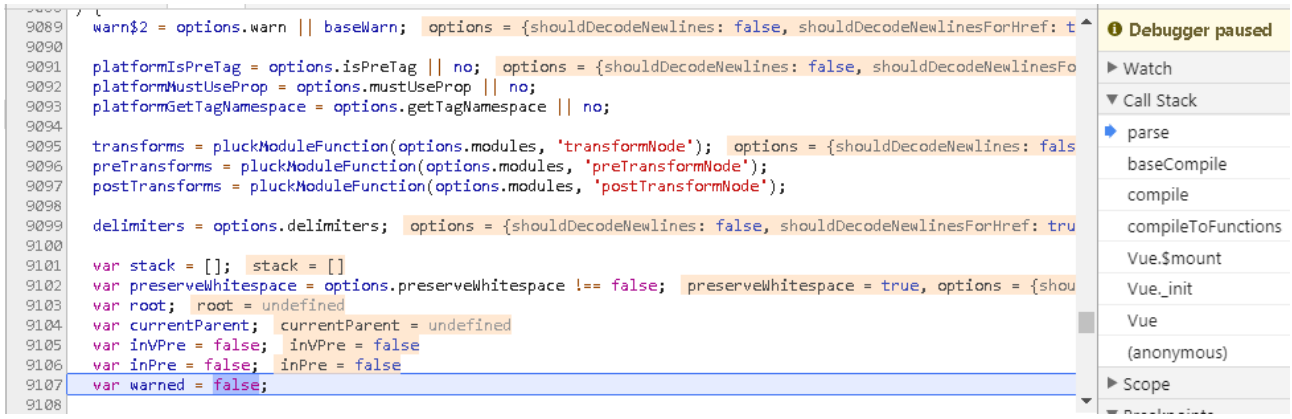
51. A função ***baseCompile*** está no contexto de chamada de ***createCompilerCreator***. É neste contexto que é chamada a função ***parse***:



VUE – Substituição de variáveis

52. Algumas outras providências:

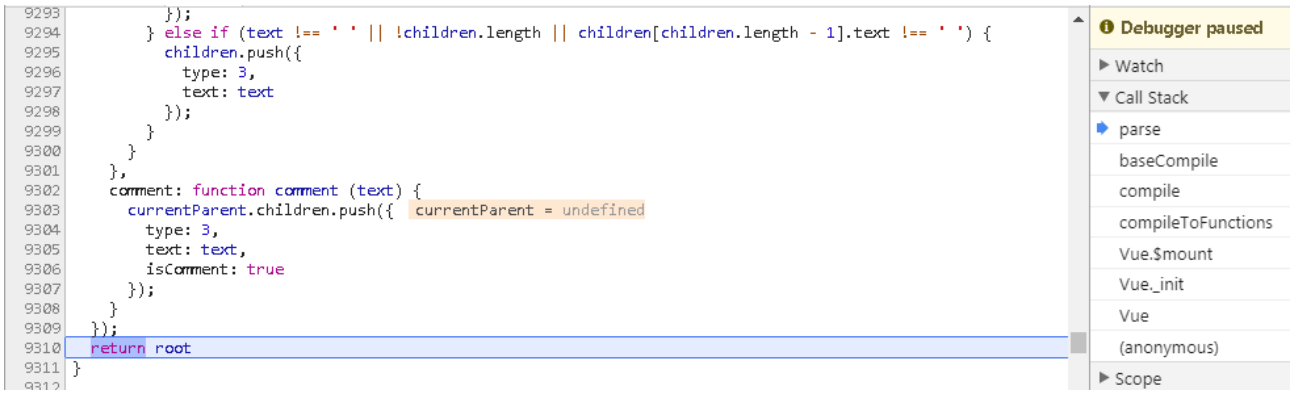
53. Acrescidas de mais algumas:



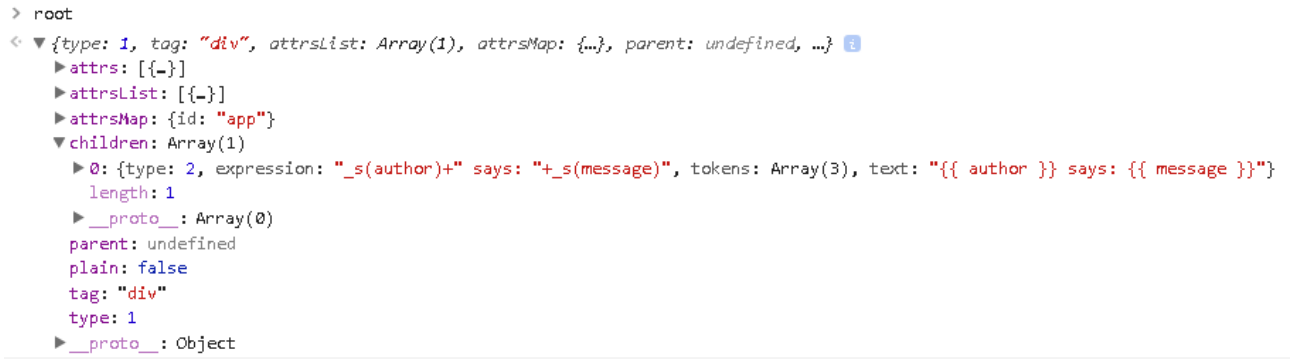
54. E agora a operação mais importante, o parseHTML, que fará a análise do contexto Vue indicado por ***"el"*** na seção **data**:

VUE – Substituição de variáveis

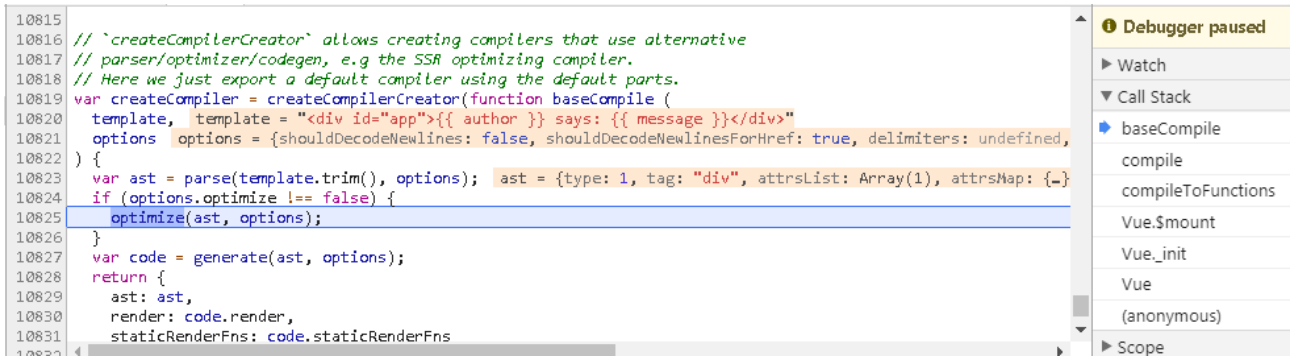
55. A função (54) retorna a variável **root**:



56. Na propriedade **children** deste objeto **root**, index 0 (zero), vemos a expressão Vue "{{ author }} says: {{ message }}" transformada no código de renderização Vue: _s(author)+" says: "+_s(message)":

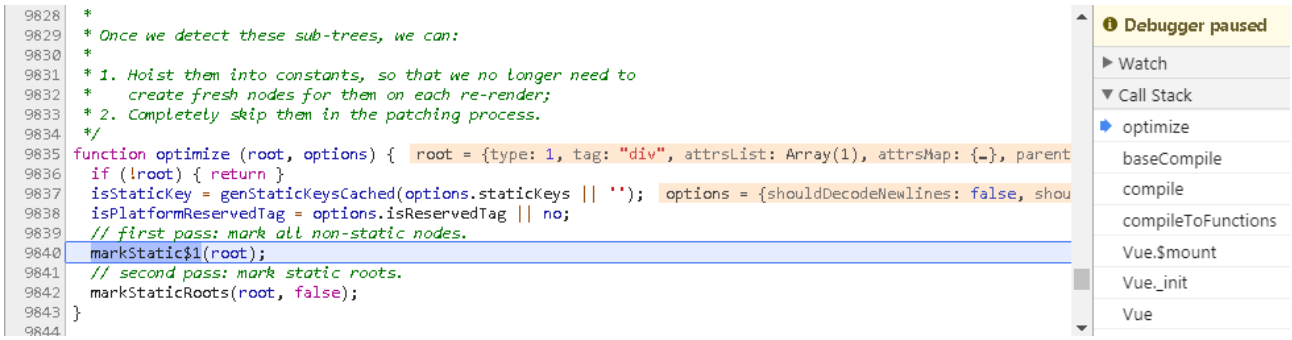


57. O código recebe alguma otimização, de acordo com as opções de compilação:

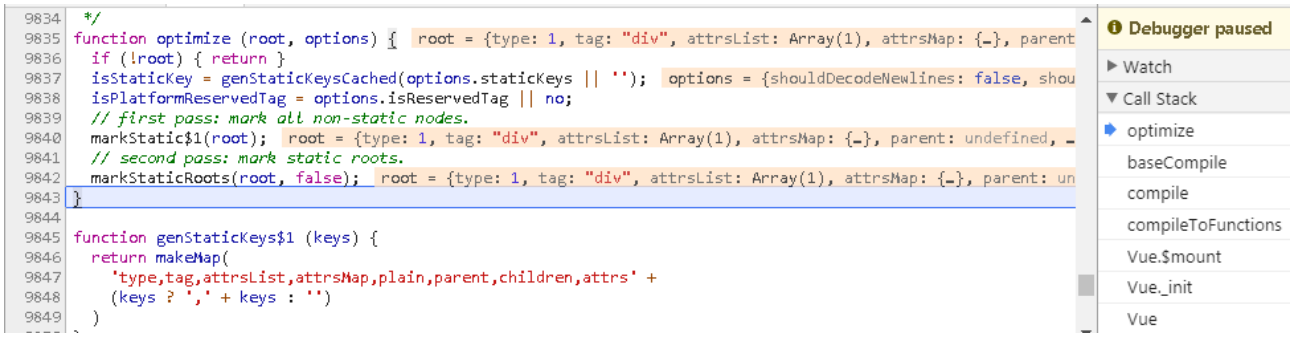


VUE – Substituição de variáveis

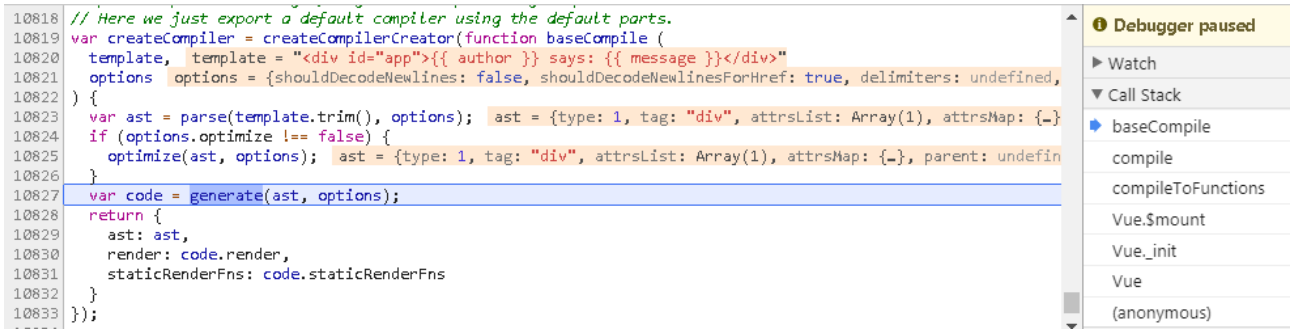
58. Esta função marca os nós não estáticos e os estáticos:



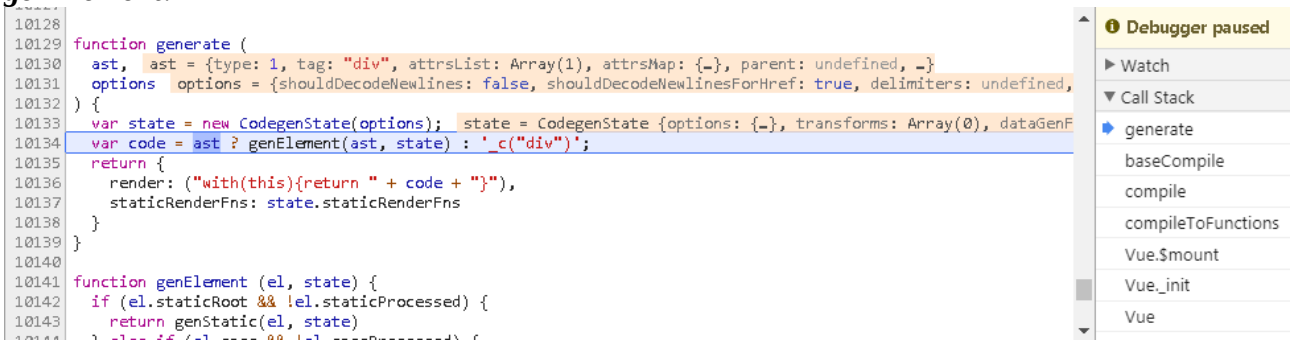
59. Os comandos terminam, e a função vai retornar à baseCompile:



60. Uma vez pronto o código da expressão Vue, será preciso gerar alguns acréscimos:



61. Como a variável **ast** possui um conteúdo (a função compilada), será chamada a função ***genElement***:



VUE – Substituição de variáveis

62. Esta função faz a checagem das diretivas que possam ter sido detectadas no elemento sendo processado (DIV), como v-for, v-once, v-if, ou se a tag é “template” ou “slot”:

```
10140
10141 function genElement (el, state) {  el = {type: 1, tag: "div", attrsList: Array(1), attrsM
10142   if (el.staticRoot && !el.staticProcessed) {
10143     return genStatic(el, state)  state = CodegenState {options: {…}, transforms: Array(0)
10144   } else if (el.once && !el.onceProcessed) {
10145     return genOnce(el, state)  state = CodegenState {options: {…}, transforms: Array(0),
10146   } else if (el.for && !el.forProcessed) {
10147     return genFor(el, state)  state = CodegenState {options: {…}, transforms: Array(0), d
10148   } else if (el.if && !el.ifProcessed) {
10149     return genIf(el, state)  state = CodegenState {options: {…}, transforms: Array(0), da
10150   } else if (el.tag === 'template' && !el.slotTarget) {
10151     return genChildren(el, state) || 'void 0'  state = CodegenState {options: {…}, transf
10152   } else if (el.tag === 'slot') {
10153     return genSlot(el, state)  state = CodegenState {options: {…}, transforms: Array(0),
10154   } else {
10155     // component or element
10156     var code;  code = undefined
10157     if (el.component) {  el = {type: 1, tag: "div", attrsList: Array(1), attrsMap: {…}, p
10158       code = genComponent(el.component, el, state);  code = undefined, state = CodegenSta
10159     } else {
10160       var data = el.plain ? undefined : genData$2(el, state);
10161
10162       var children = el.inlineTemplate ? null : genChildren(el, state, true);
10163       code = "_c('" + (el.tag) + "'" + (data ? "," + data) : '') + (children ? "," + ch
10164     }
10165     // module transforms
```

63. Na linha 10160, é chamada a função ***genData$2***:

```
10276 function genData$2 (el, state) {  el = {type: 1, tag: "div", attrsList: Array(1), attrsMap: {…}, parent: und
10277   var data = '{';
10278
10279   // directives first.
10280   // directives may mutate the el's other properties before they are generated.
10281   var dirs = genDirectives(el, state);
10282   if (dirs) { data += dirs + ','; }
10283
10284   // key
10285   if (el.key) {
10286     data += "key:" + (el.key) + ",";
10287   }
10288   // ref
10289   if (el.ref) {
10290
{}  Line 10277, Column 14
```

▶ Watch
▼ Call Stack
genData$2
genElement
generate
baseCompile
compile
compileToFunctions
Vue.$mount
Vue._init
Vue

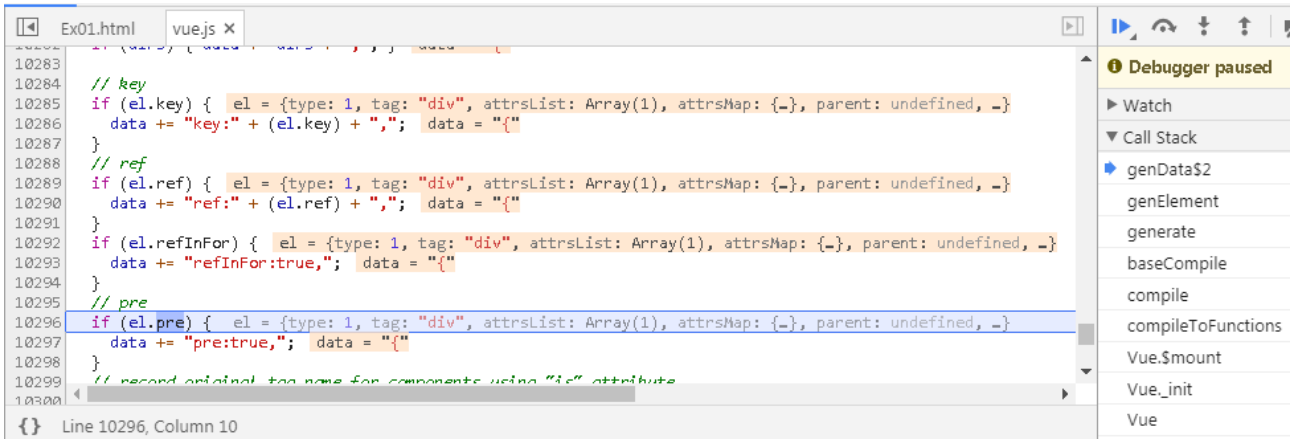
64. E na linha 10281 de ***genData$2*** é chamada a função ***genDirectives***:

```
10352 }
10353
10354 function genDirectives (el, state) {  el = {type: 1, tag: "div", attrsList: Array(1), attrsMap: {…}, parent:
10355   var dirs = el.directives;  dirs = undefined
10356   if (!dirs) { return }
10357   var res = 'directives:[';
10358   var hasRuntime = false;
10359   var i, l, dir, needRuntime;
10360   for (i = 0, l = dirs.length; i < l; i++) {
10361     dir = dirs[i];
10362     needRuntime = true;
10363     var gen = state.directives[dir.name];
10364     if (gen) {
10365       // compile-time directive that manipulates AST.
10366       // returns true if it also needs a runtime counterpart.
10367       needRuntime = !!gen(el, dir, state.warn);
10368     }
10369
{}  Line 10356, Column 22
```

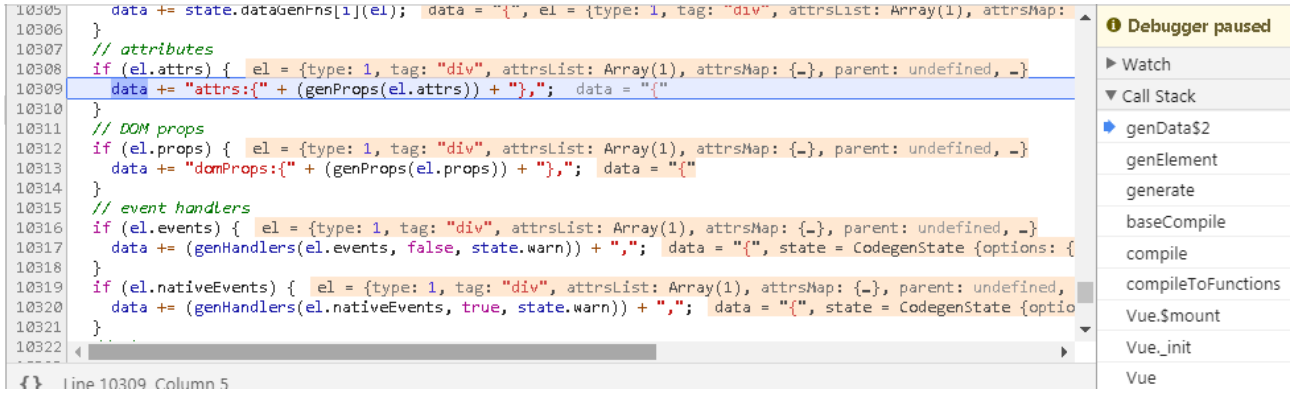
Debugger paused
▶ Watch
▼ Call Stack
genDirectives
genData$2
genElement
generate
baseCompile
compile
compileToFunctions
Vue.$mount
Vue._init

VUE – Substituição de variáveis

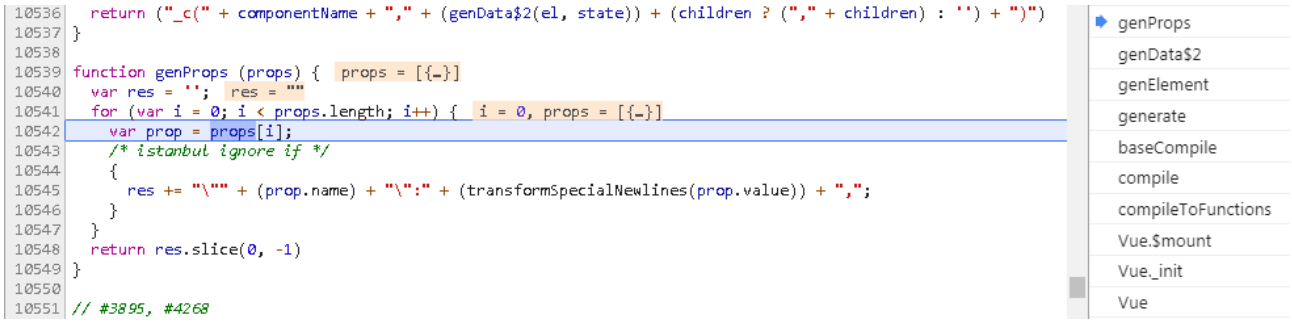
65. Ainda são testados outros tipos de diretivas:



66. E também atributos. Estes estão presentes em nosso exemplo:

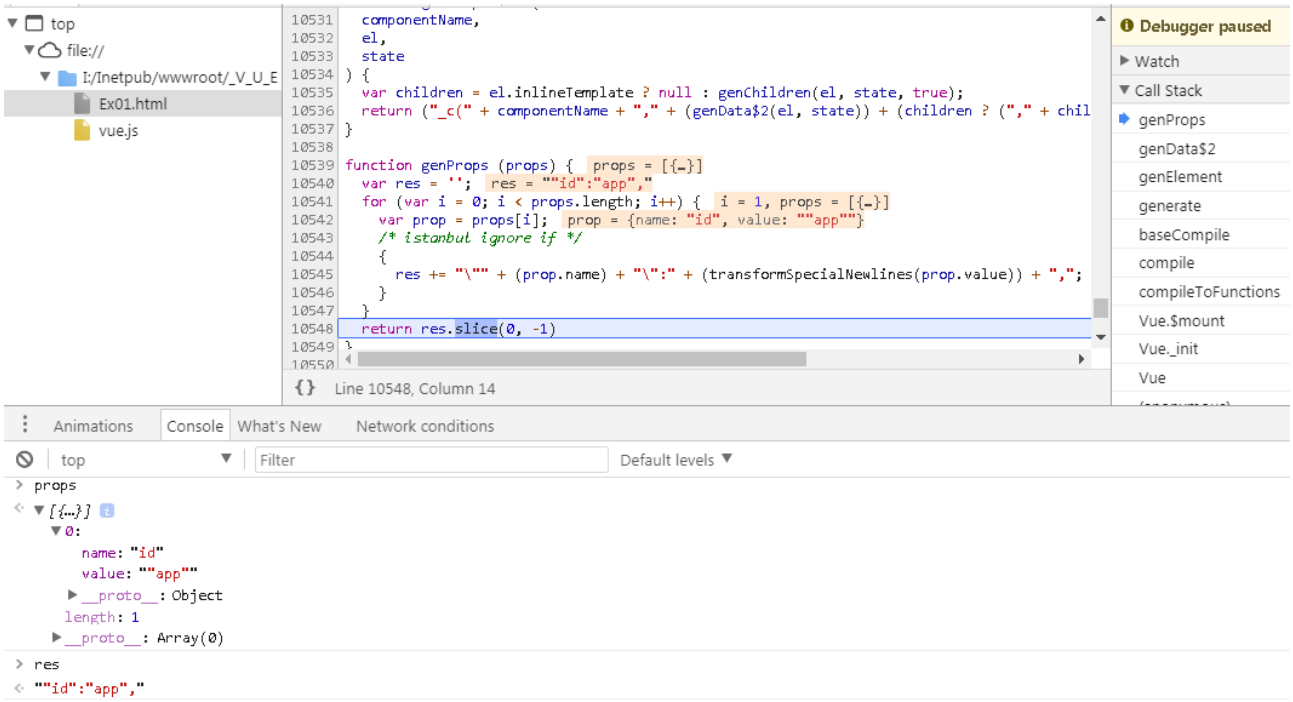


67. A função ***genData$2*** chama ***genProps*** com o parâmetro **props**, que contém todos os atributos encontrados na tag em análise:

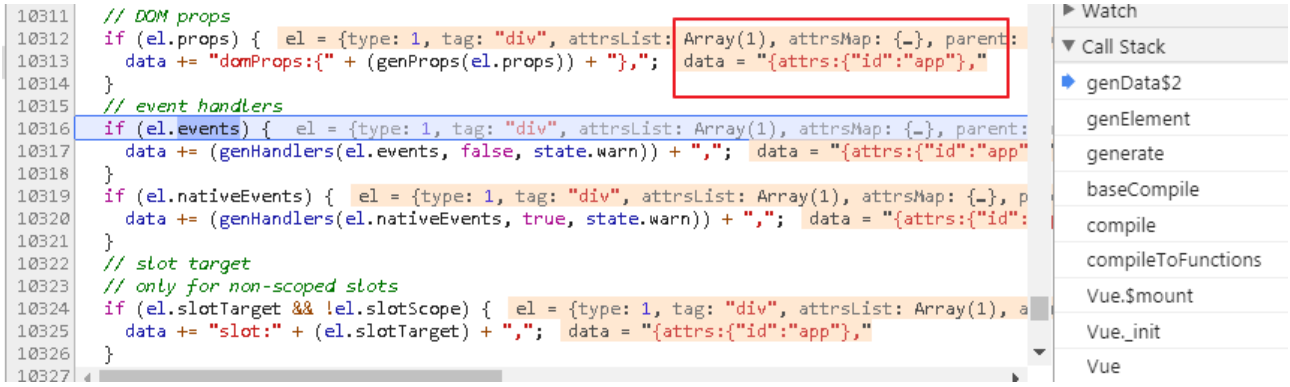


VUE – Substituição de variáveis

68. A figura a seguir mostra a única propriedade a ser aplicada: o atributo “**id**” com valor “**app**”.

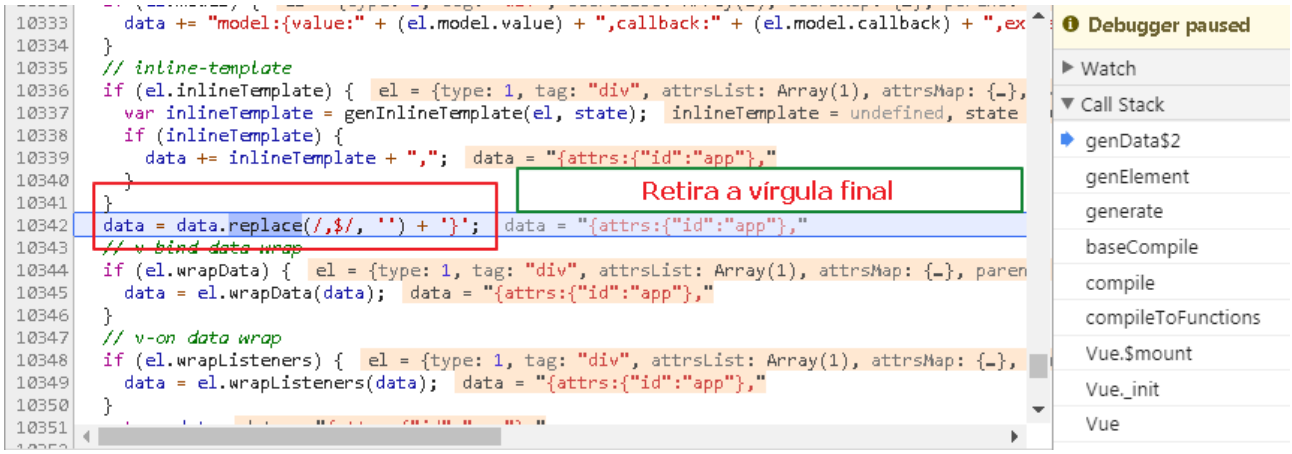


69. A variável data recebe um objeto tipo JSON (incompleto) com os atributos detectados:

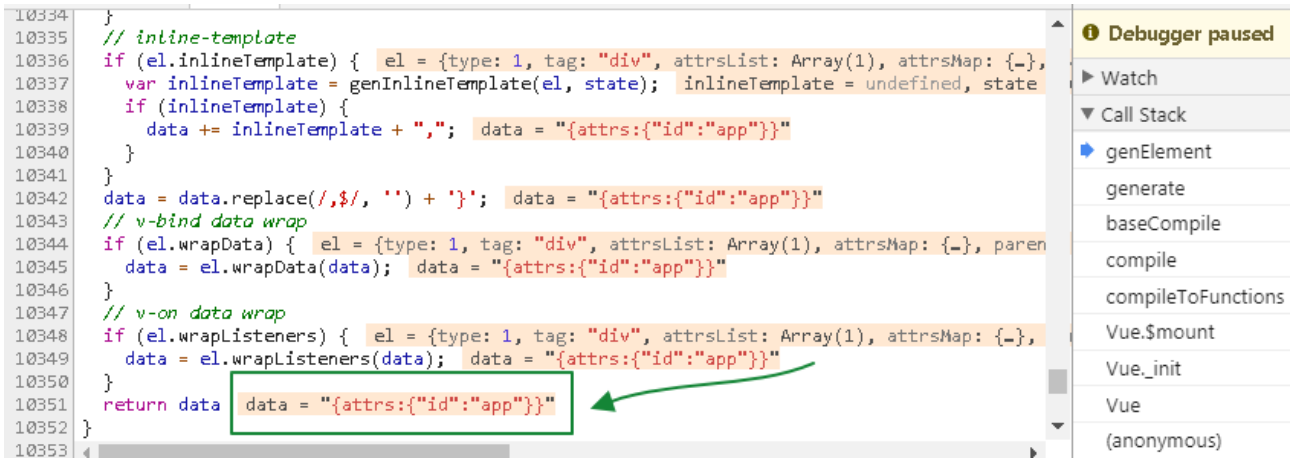


VUE – Substituição de variáveis

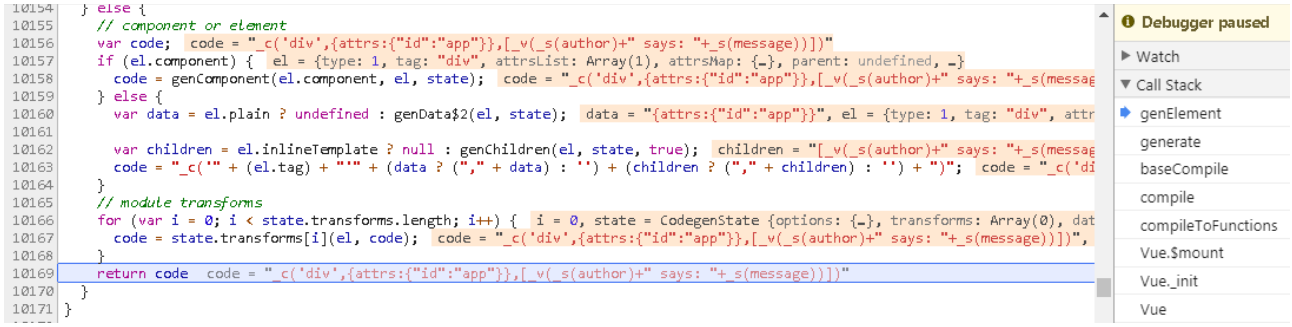
70. Ao final das propriedades, a última vírgula é extraída através de uma expressão regular:



71. E o resultado recebe o fecha chaves "}":

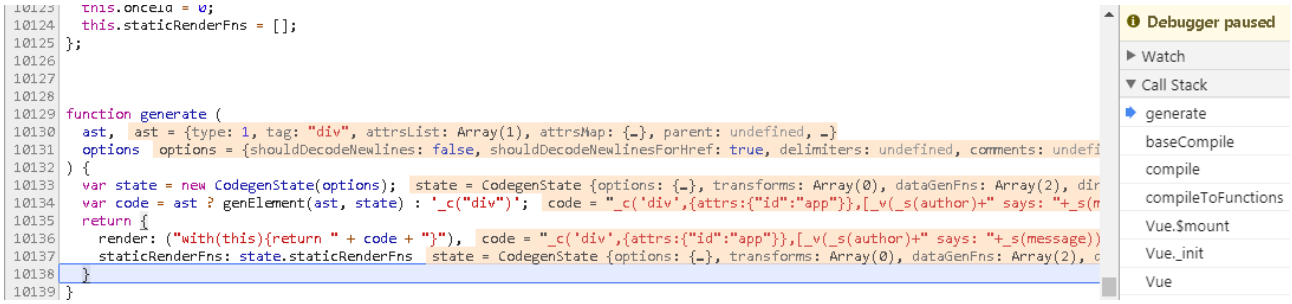


72. Os atributos são acrescentados à função de renderização, como mostrado a seguir:

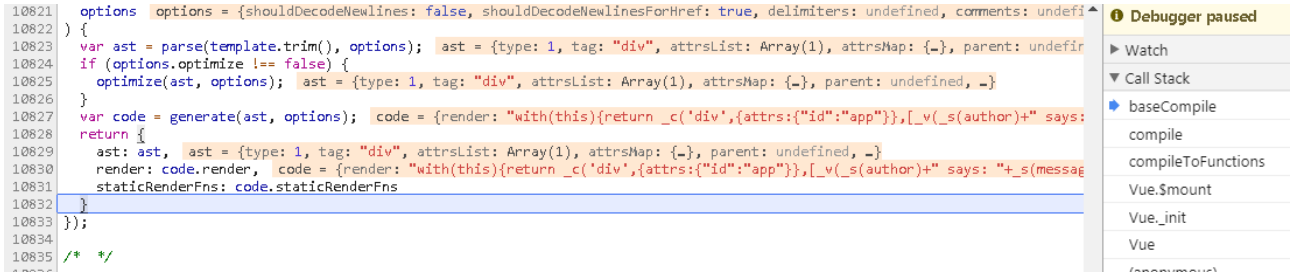


VUE – Substituição de variáveis

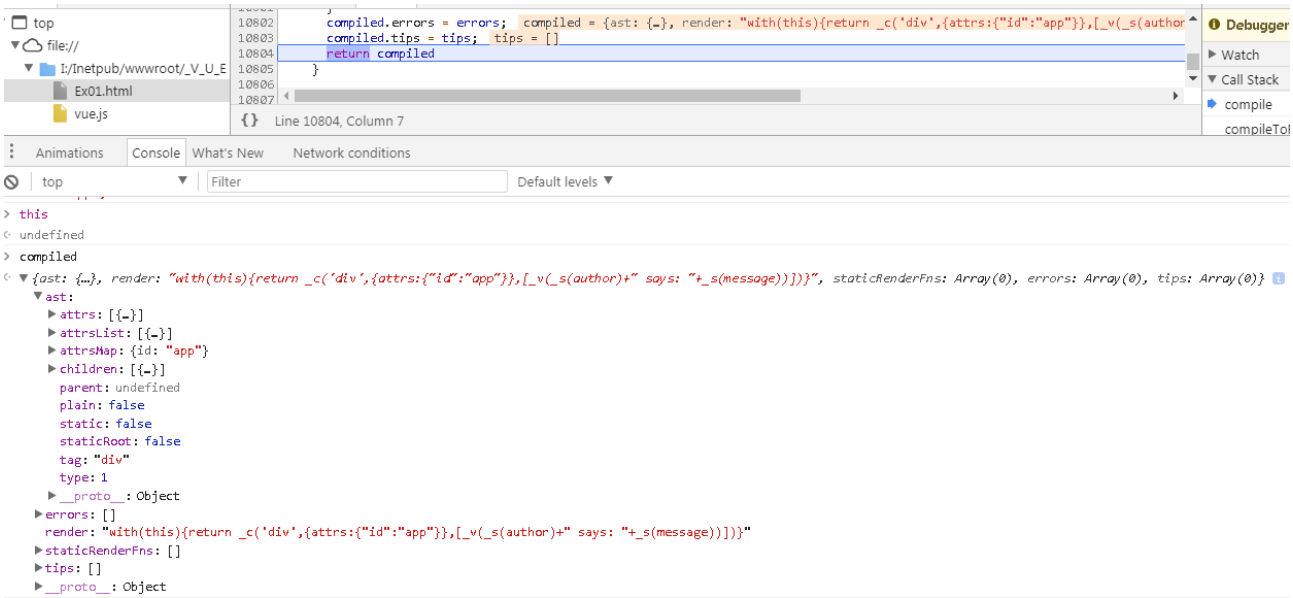
73. O javascript retorna à função ***generate***:



74. O javascript retorna à função ***baseCompile***:

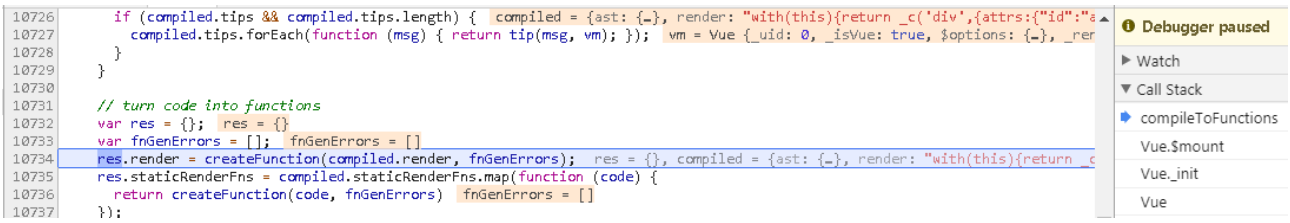


75. Abaixo o objeto **compiled** retornado pelo javascript:

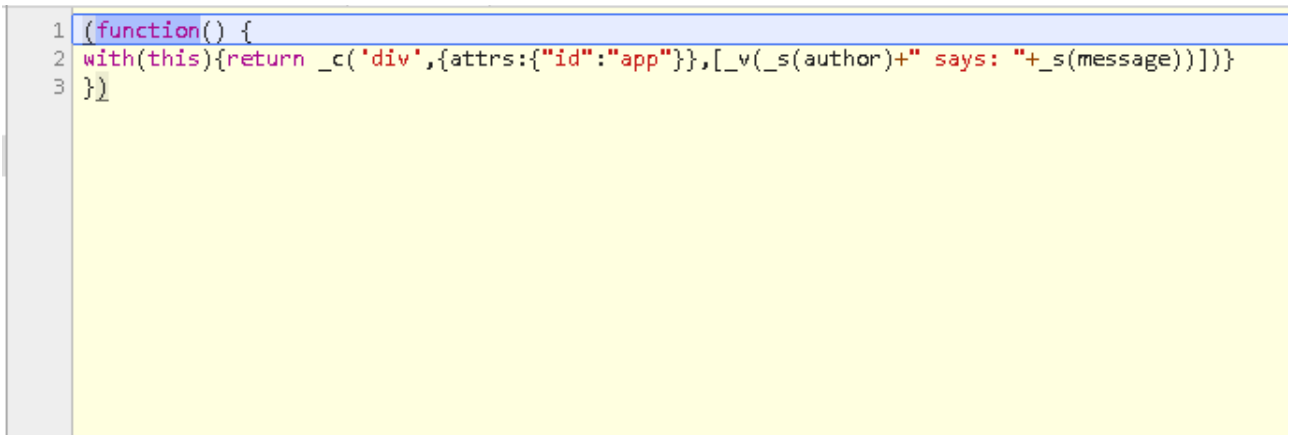


VUE – Substituição de variáveis

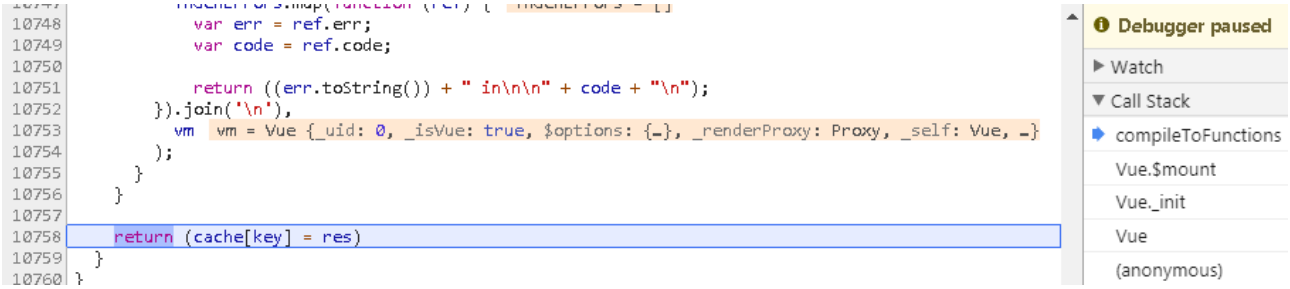
76. O javascript retorna à função ***compileToFunctions***:



77. Este é o produto da compilação do código HTML, chamado de função de renderização:

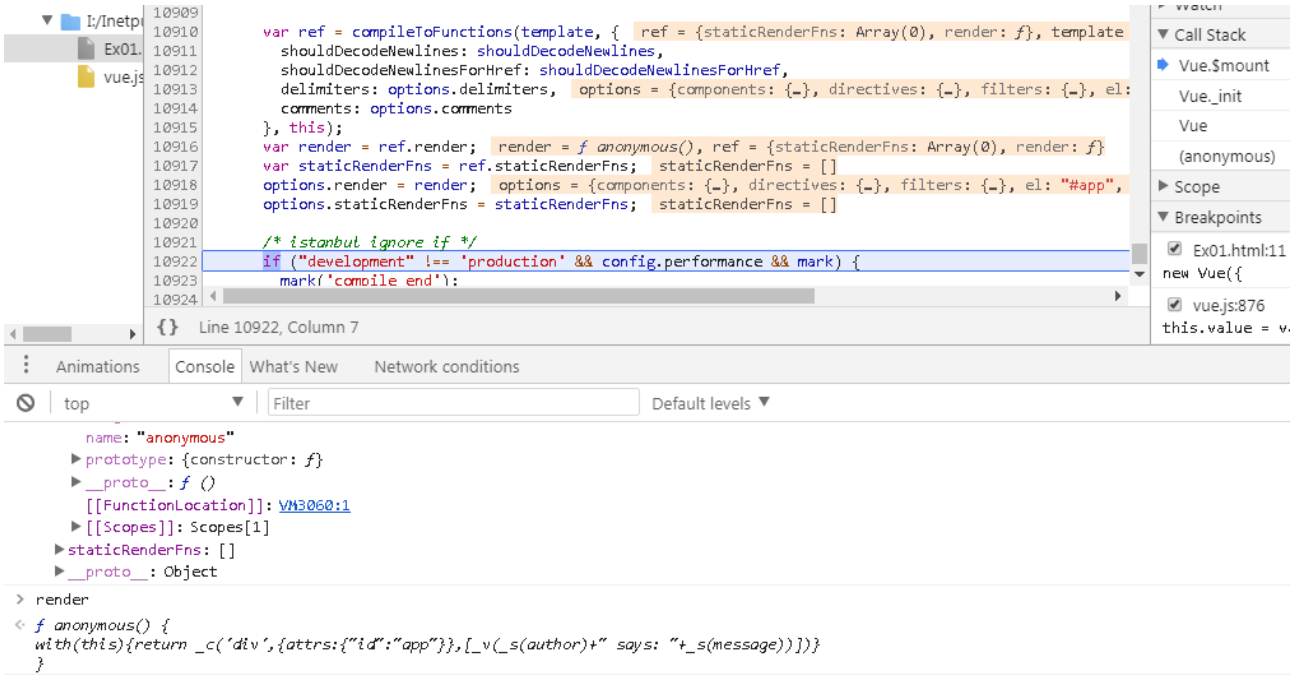


78. Chegamos à última linha de compileToFunctions. A função vai retornar para Vue._init.

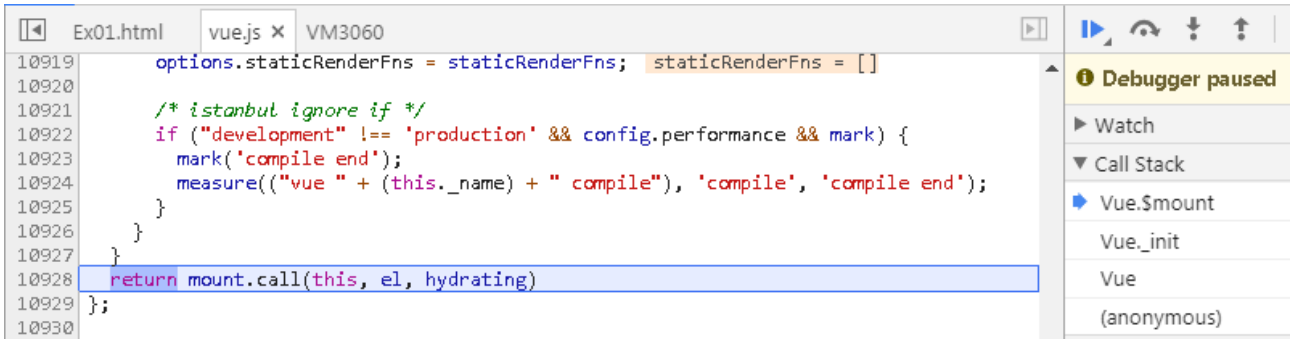


VUE – Substituição de variáveis

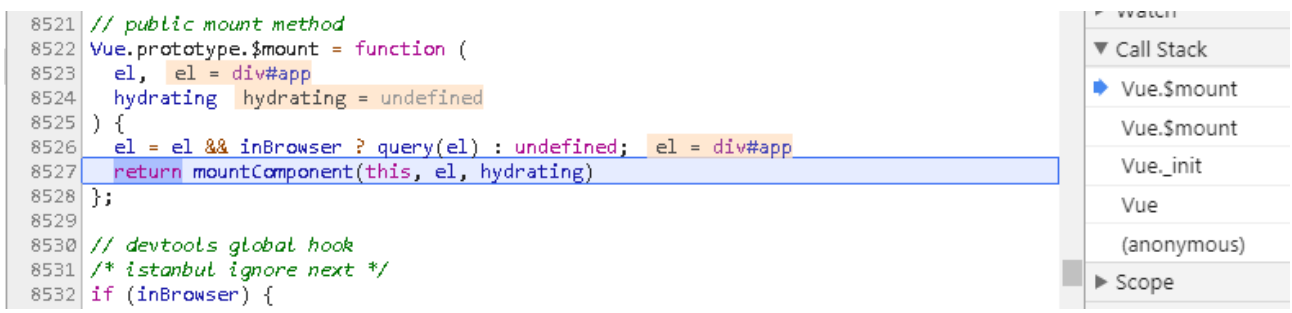
79. A parte inferior do debug mostra a função render obtida pela função de compilação:



80. Mas uma vez preparada a função de renderização, ela precisa ser executada. A função que começa a providenciar esta ação é Vue.$mount:

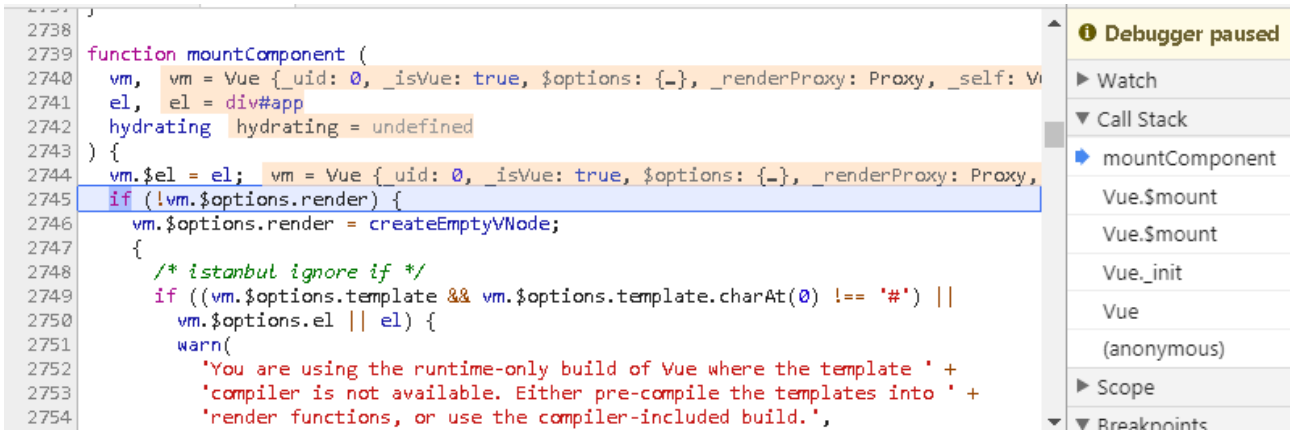


81. Esta função coleta o elemento de contexto Vue (div#app) e chama por mountComponent. Parece a montagem de um componente, e realmente é. Quando o componente está no próprio corpo do HTML, esta função é chamada normalmente, bastando apenas indicar o “ponto de montagem” declarado na seção “**el**”.

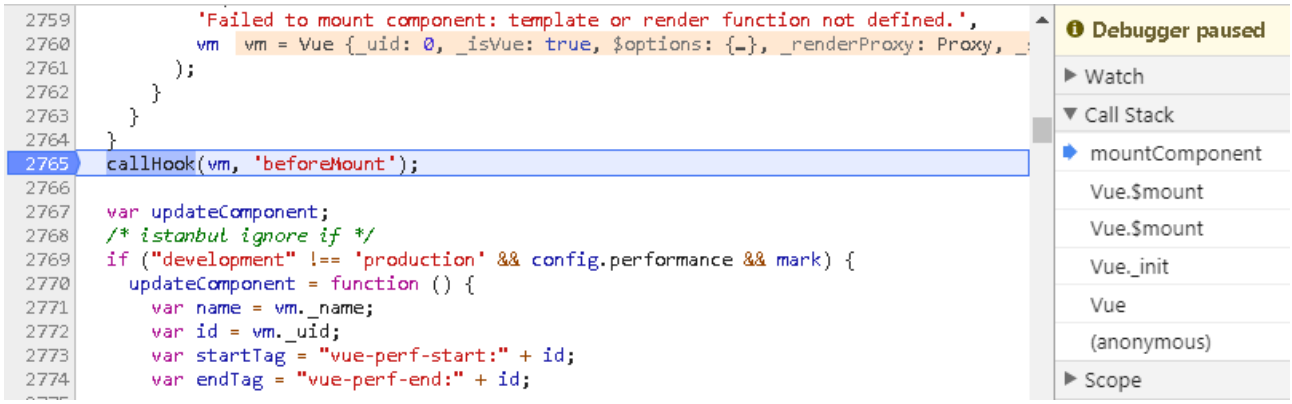


VUE – Substituição de variáveis

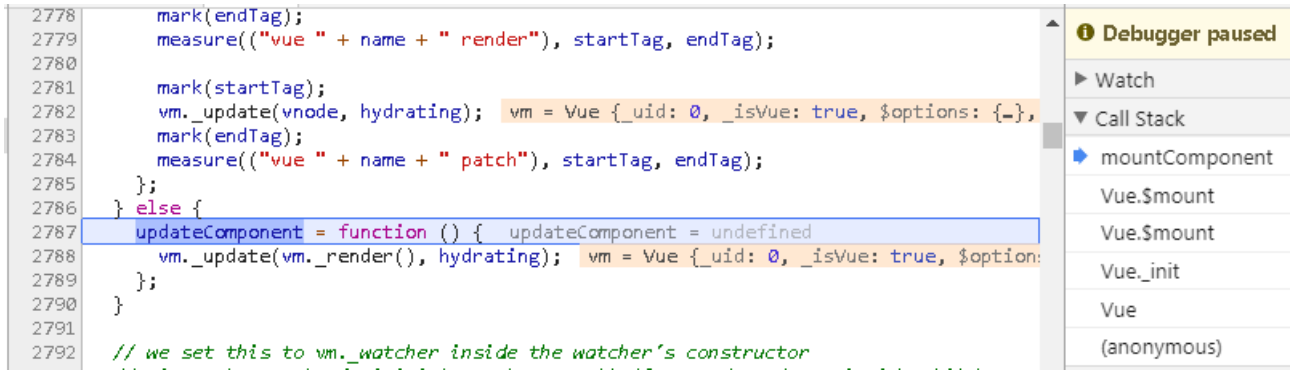
82. A função testa se existe uma função render. Se não houver, cria um nó vazio, o que não é nosso caso.



83. É chamado o “hook” cujo identificador é “***beforeMount***”. Hooks são como eventos do objeto Vue.

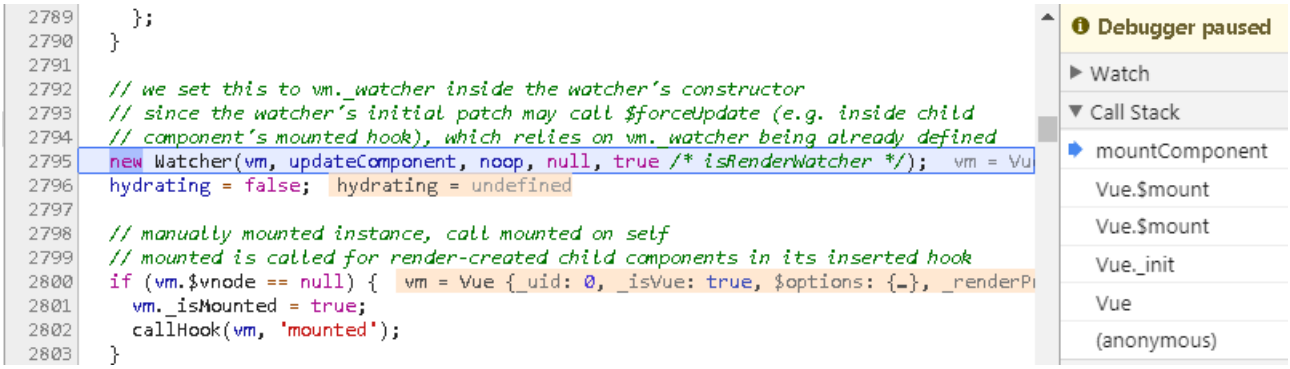


84. A próxima chamada é a da função updateComponent:

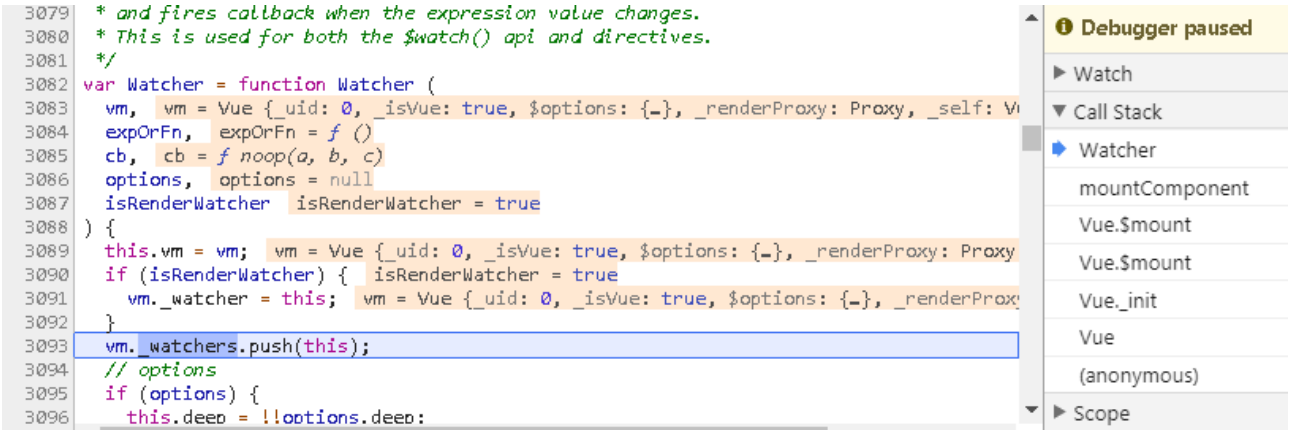


VUE – Substituição de variáveis

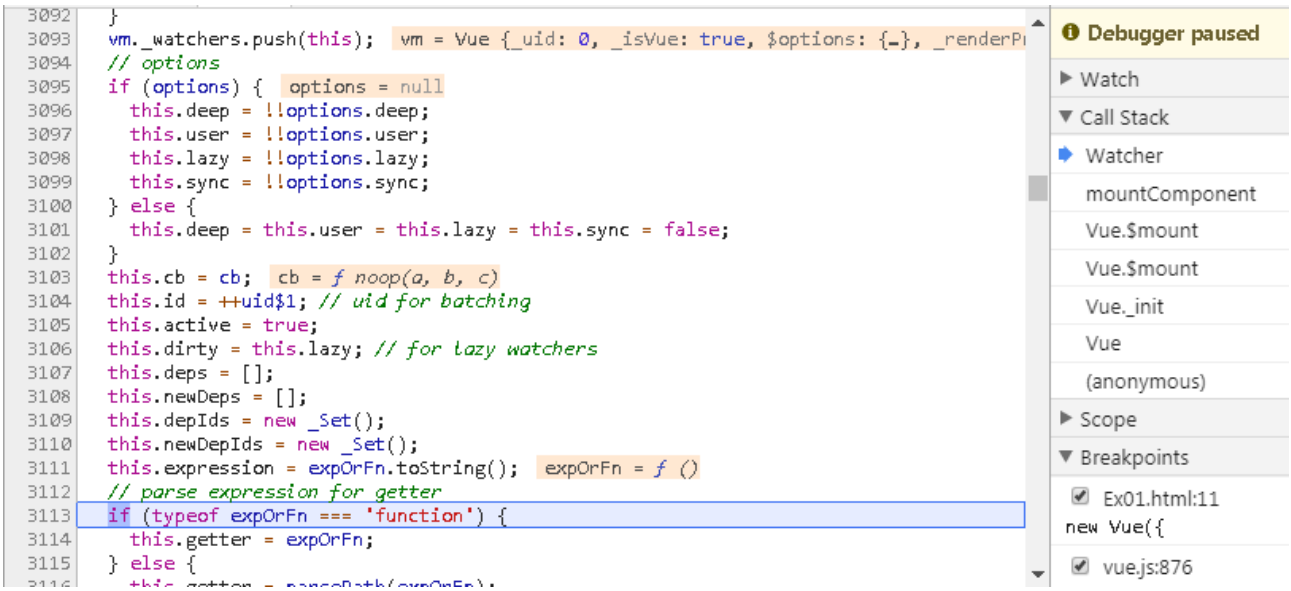
85. Para possibilitar o monitoramento do “componente” estabelecido pelo Vue, estanque ou incorporado, é preciso criar um objeto Watcher no corpo do Vue:



86. Aqui, vemos o objeto Watcher sendo inserido no corpo do objeto Vue:

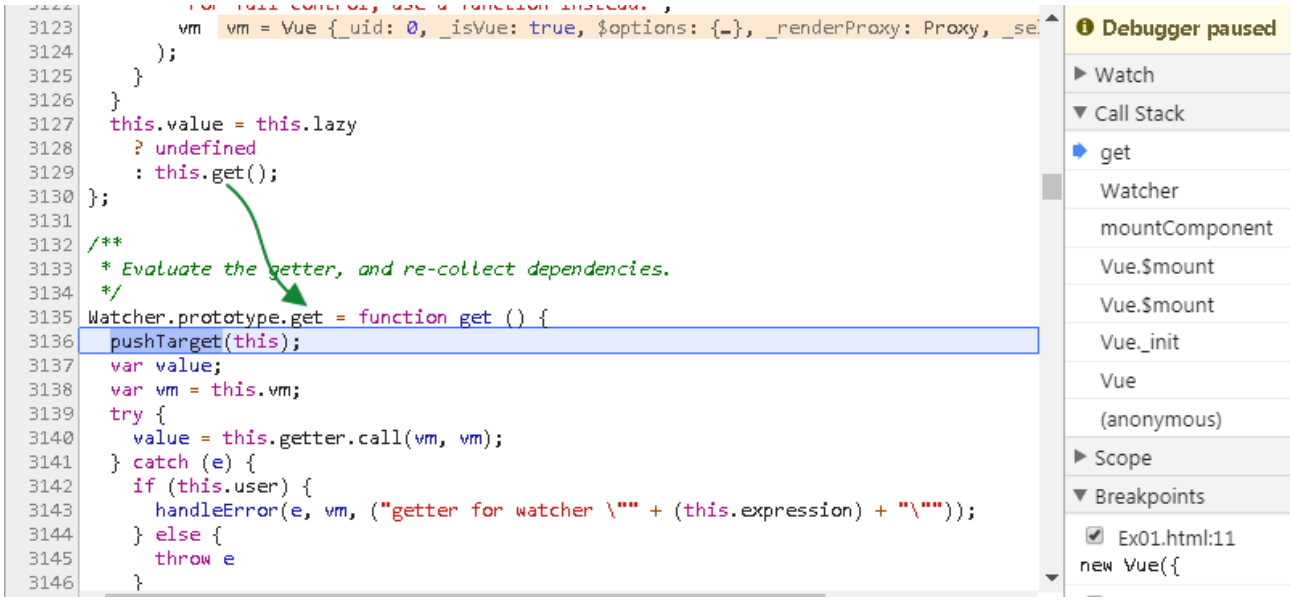


87. Seguem outras definições do objeto Watcher, com a declaração do método getter:



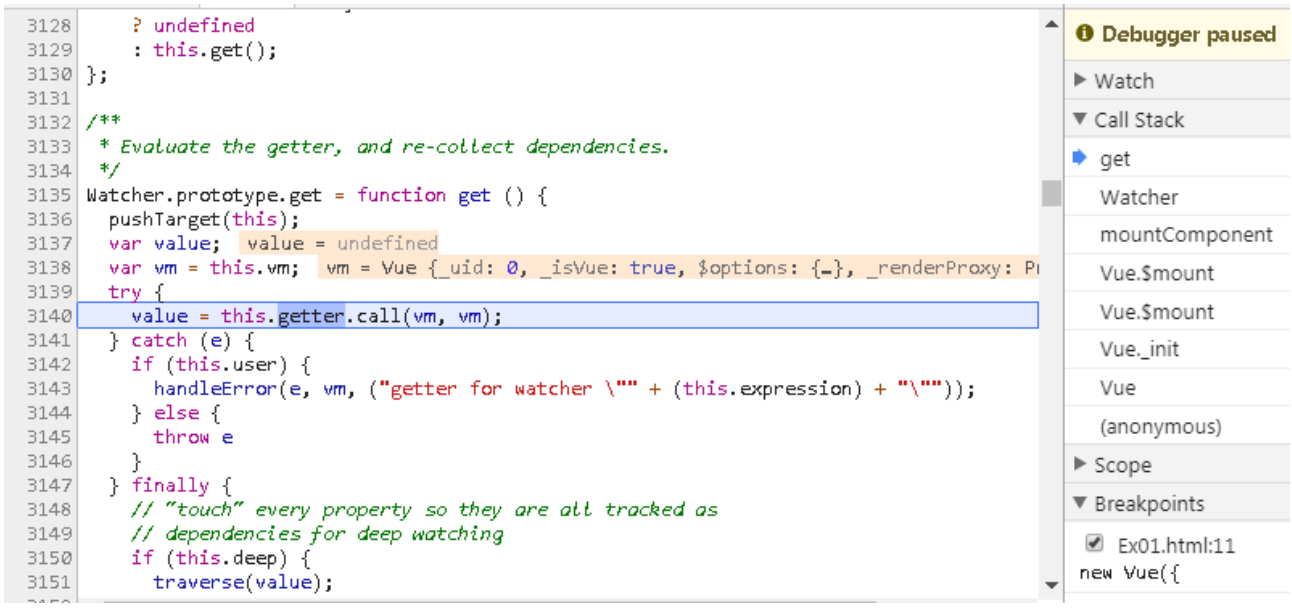
VUE – Substituição de variáveis

88. O objeto **Watcher** recebe o seu método **get**:




```
3123        vm  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _se
3124      );
3125    }
3126  }
3127  this.value = this.lazy
3128    ? undefined
3129    : this.get();
3130 };
3131
3132 /**
3133  * Evaluate the getter, and re-collect dependencies.
3134  */
3135 Watcher.prototype.get = function get () {
3136   pushTarget(this);
3137   var value;
3138   var vm = this.vm;
3139   try {
3140     value = this.getter.call(vm, vm);
3141   } catch (e) {
3142     if (this.user) {
3143       handleError(e, vm, ("getter for watcher \"" + (this.expression) + "\""));
3144     } else {
3145       throw e
3146     }
```

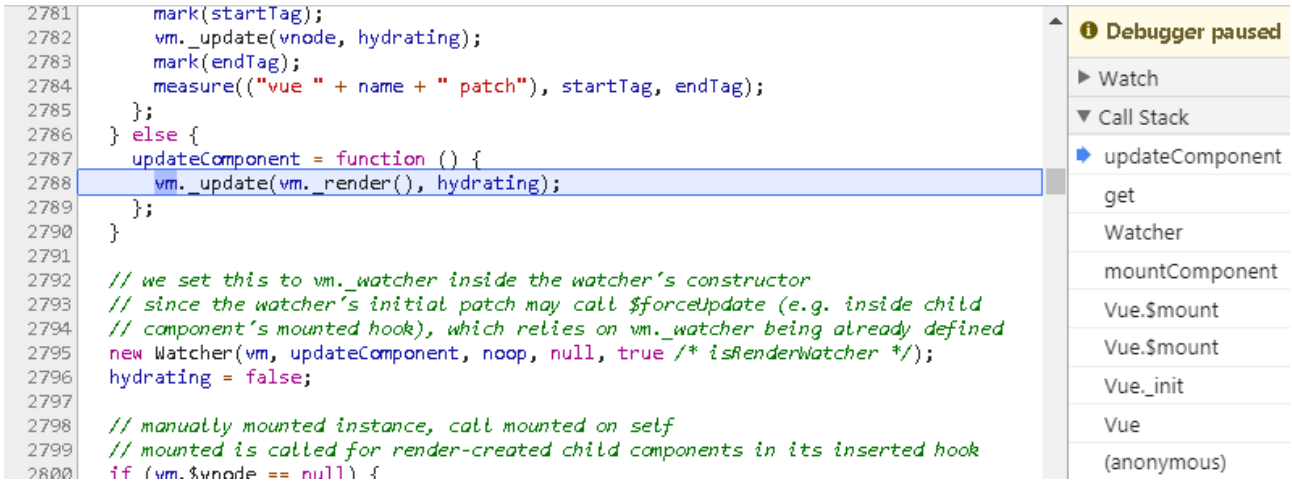
89. O método **get** é chamado. O getter do objeto Watcher reside na função updateComponent:



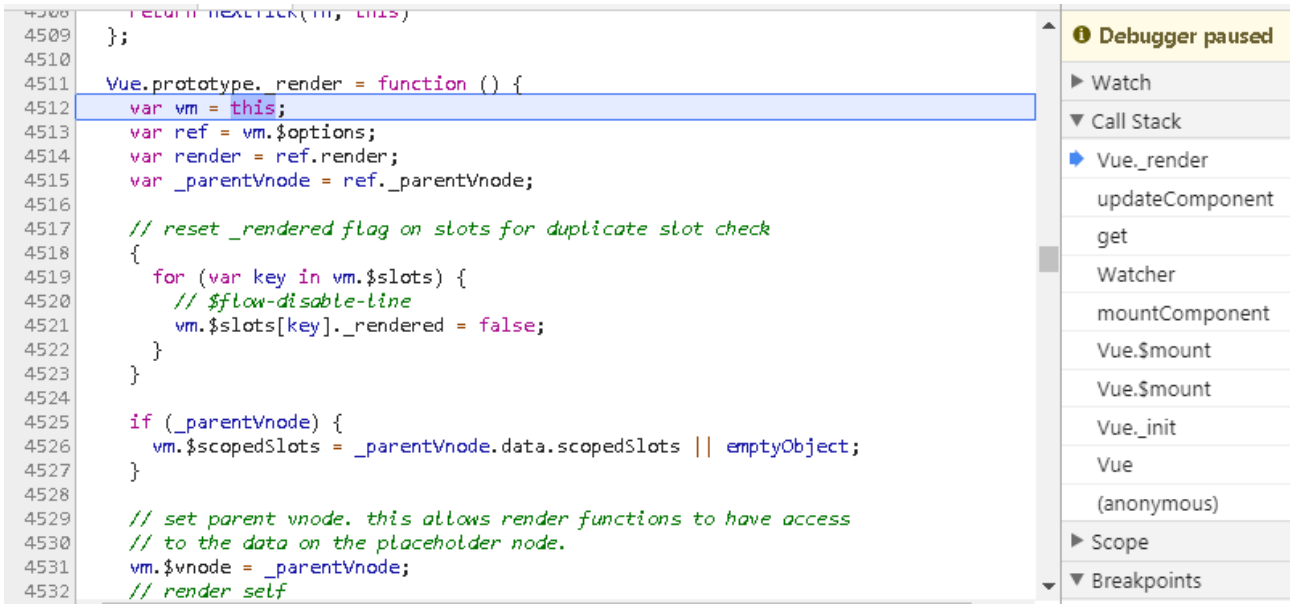

```
3128    ? undefined
3129    : this.get();
3130 };
3131
3132 /**
3133  * Evaluate the getter, and re-collect dependencies.
3134  */
3135 Watcher.prototype.get = function get () {
3136   pushTarget(this);
3137   var value;  value = undefined
3138   var vm = this.vm;  vm = Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: P
3139   try {
3140     value = this.getter.call(vm, vm);
3141   } catch (e) {
3142     if (this.user) {
3143       handleError(e, vm, ("getter for watcher \"" + (this.expression) + "\""));
3144     } else {
3145       throw e
3146     }
3147   } finally {
3148     // "touch" every property so they are all tracked as
3149     // dependencies for deep watching
3150     if (this.deep) {
3151       traverse(value);
```

VUE – Substituição de variáveis

90. A função updateComponent reside na linha 2788 de Vue.js.

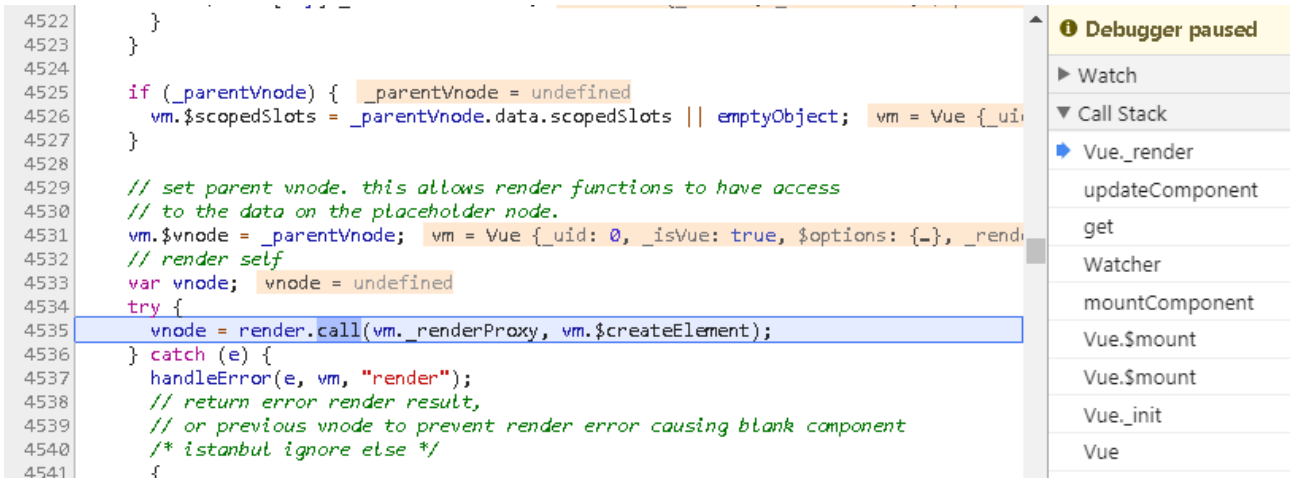


91. O primeiro parâmetro desta função é obtido pelo método **Vue._render**:

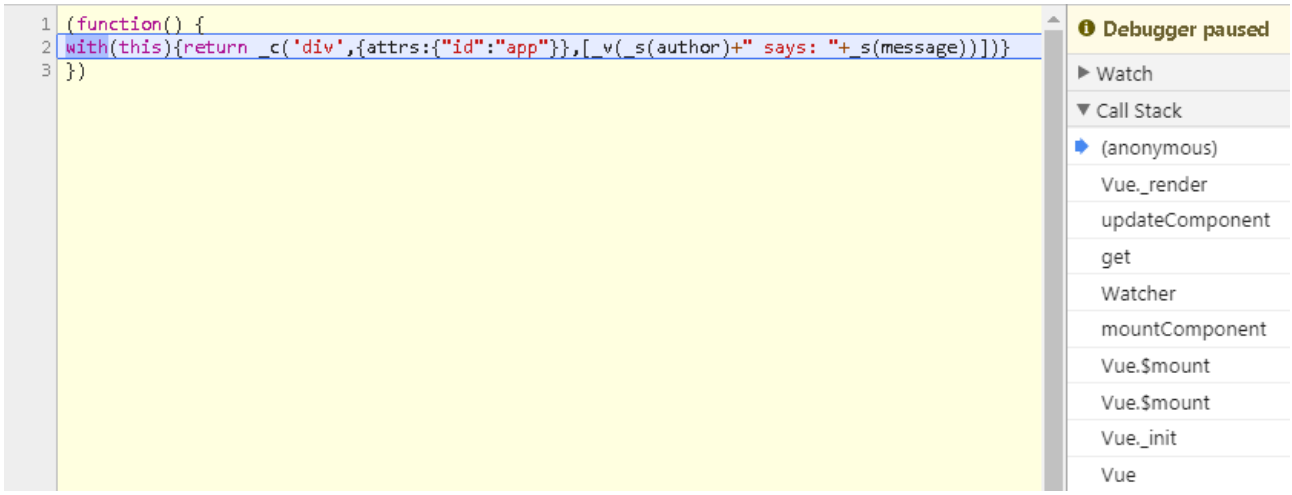


VUE – Substituição de variáveis

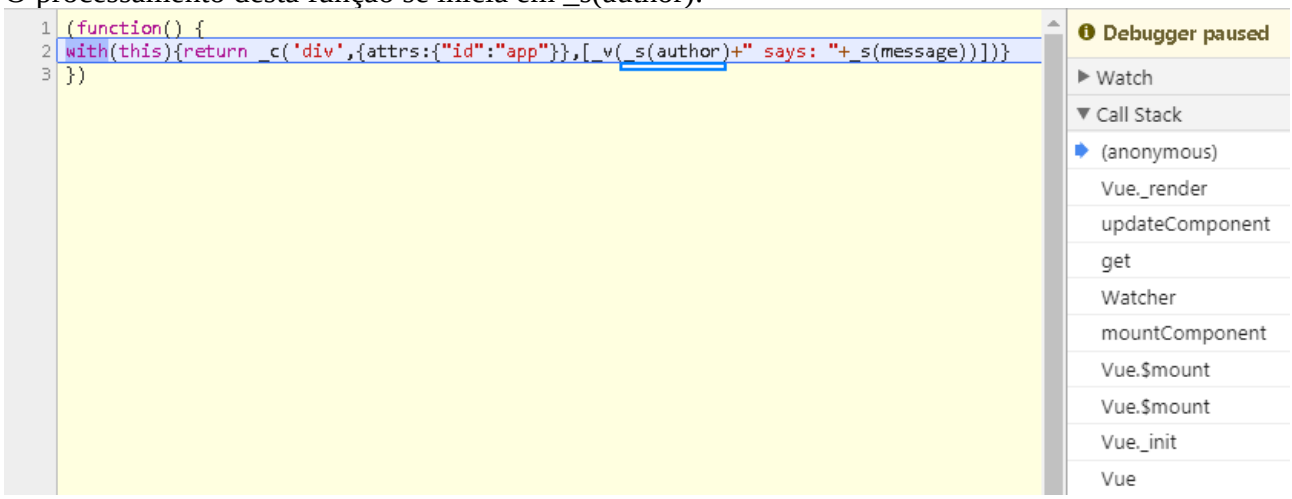
92. O processamento prossegue, e algumas linhas abaixo é executada a função **render**:



93. E **render** se refere à função gerada no processo de compilação:

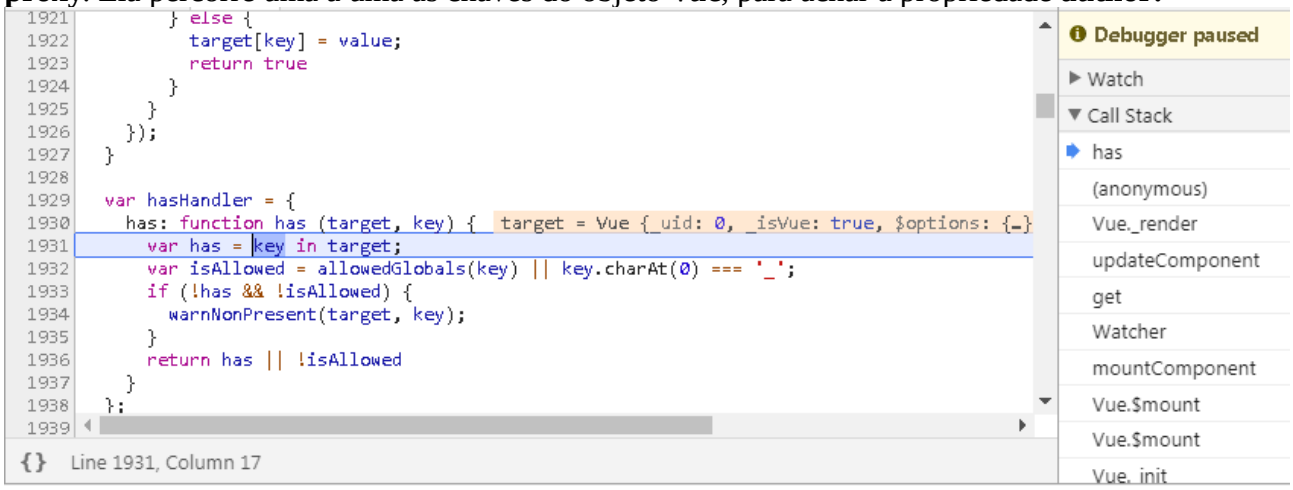


O processamento desta função se inicia em _s(author):

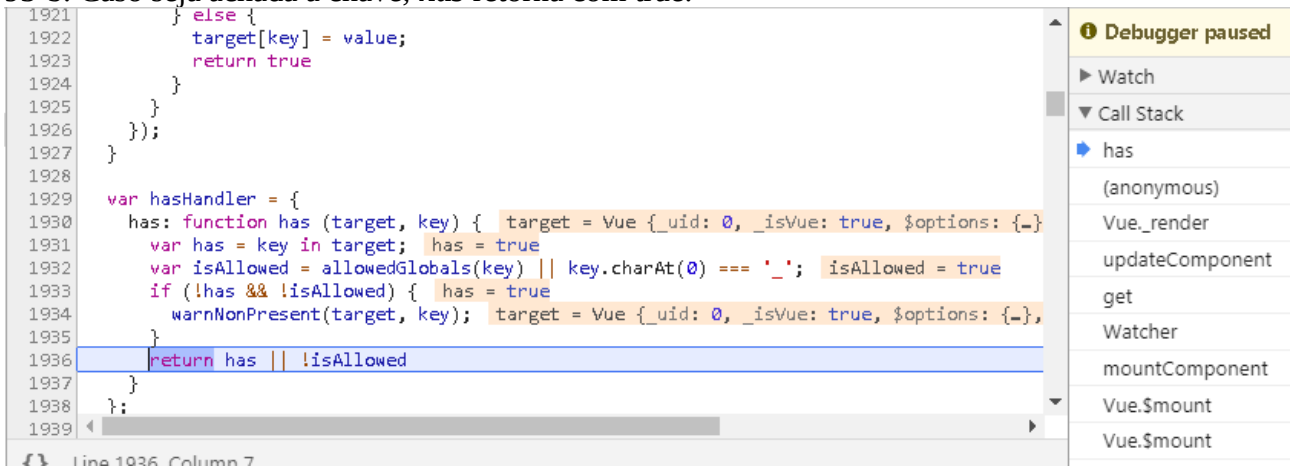


VUE – Substituição de variáveis

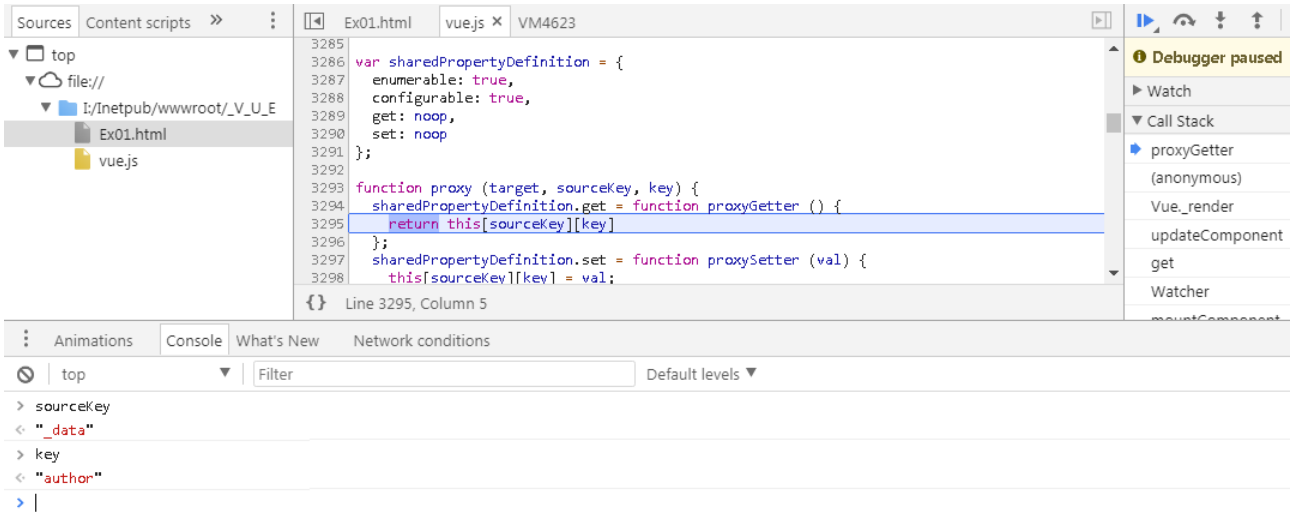
93-a. Com **author** como propriedade (não como conteúdo), é chamada a função ***has*** do objeto **proxy**. Ela percorre uma a uma as chaves do objeto **Vue**, para achar a propriedade **author:**



93-b. Caso seja achada a chave, ***has*** retorna com true:

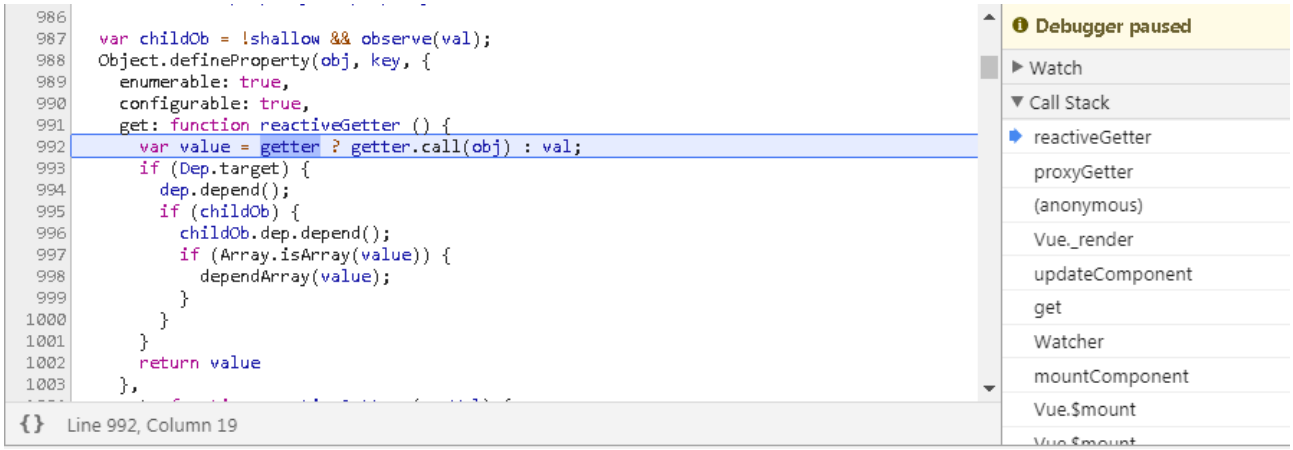


93-c. O valor é retornado pelo método ***get*** do objeto **proxy**, contido na propriedade/array "**_data**", no item cuja chave é "**author**":

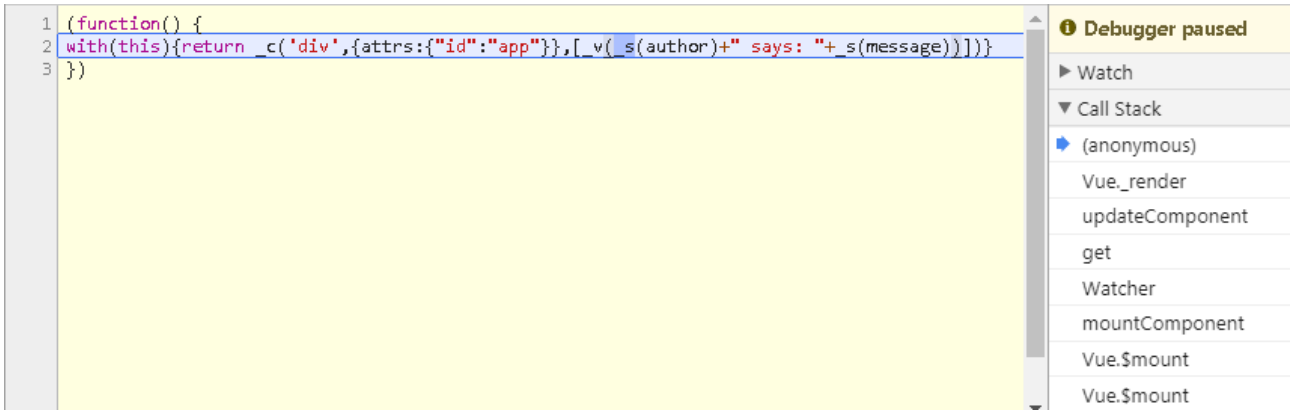


VUE – Substituição de variáveis

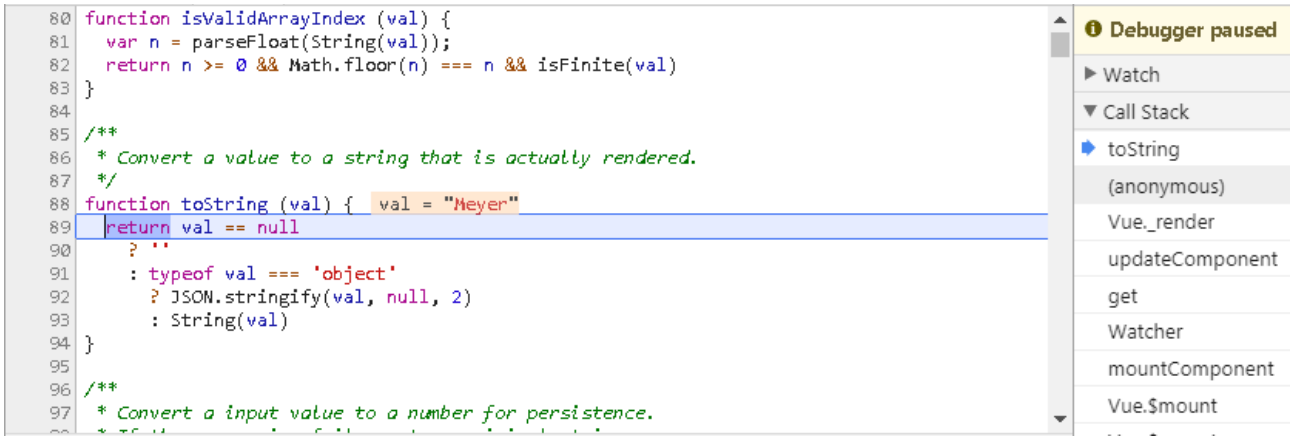
93-d. O método ***get*** de **proxy** invoca o método ***get*** de ***reactiveGetter***, definido em ***defineReactive***, para verificar se não existe um outro ***getter*** dedicado ao objeto **proxy** definido pelo usuário:



93-e. ***Vue._render*** retorna à função de renderização no ponto onde foi chamada:

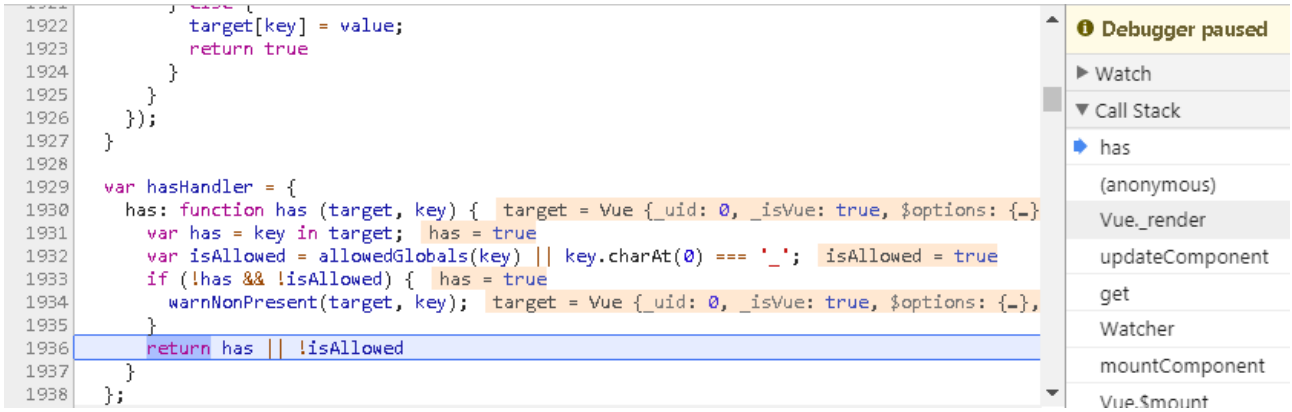


93-f. O valor constante na variável proxy (ou gerenciada por proxy) é convertido pela função **_s(…)** que invoca ***toString***:

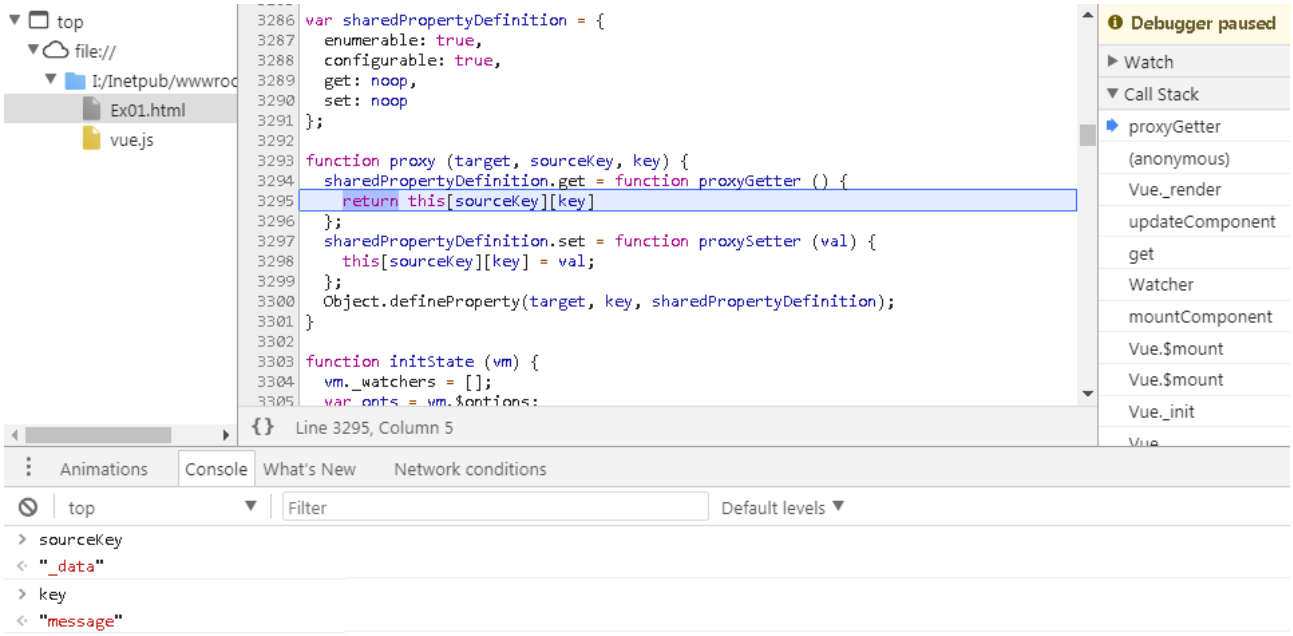


VUE – Substituição de variáveis

93-g. Posteriormente se dá o retorno por ***has*** do objeto Proxy:

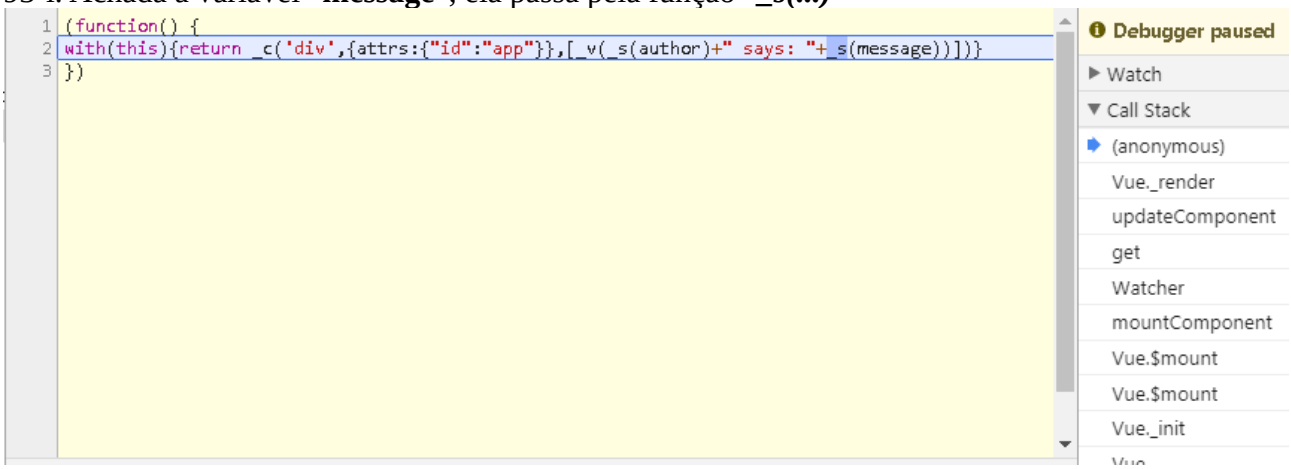


93-h. O próximo processamento se refere à variável "**message**":

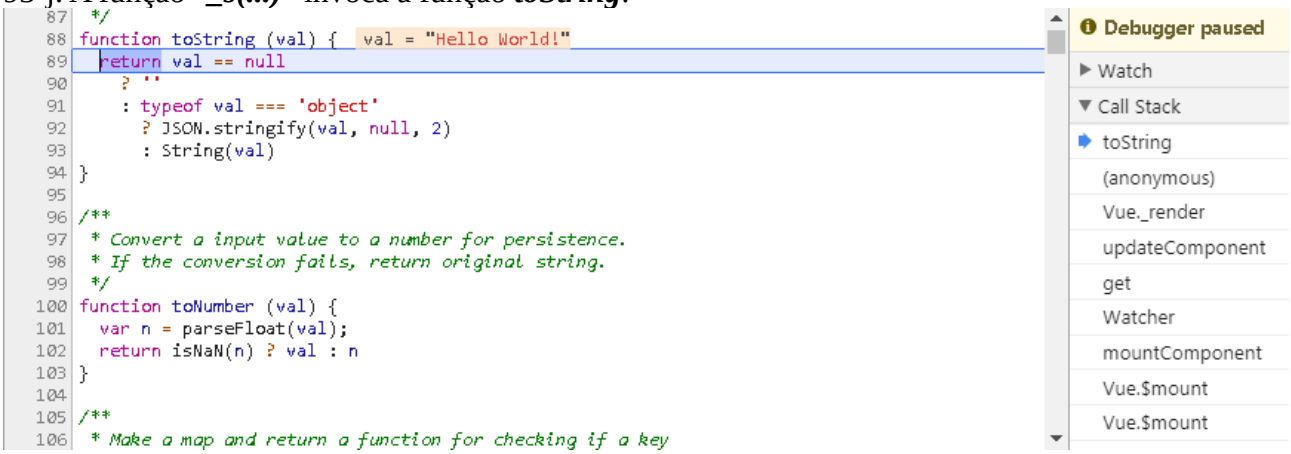


VUE – Substituição de variáveis

93-i. Achada a variável "**message**", ela passa pela função "***_s(...)***"

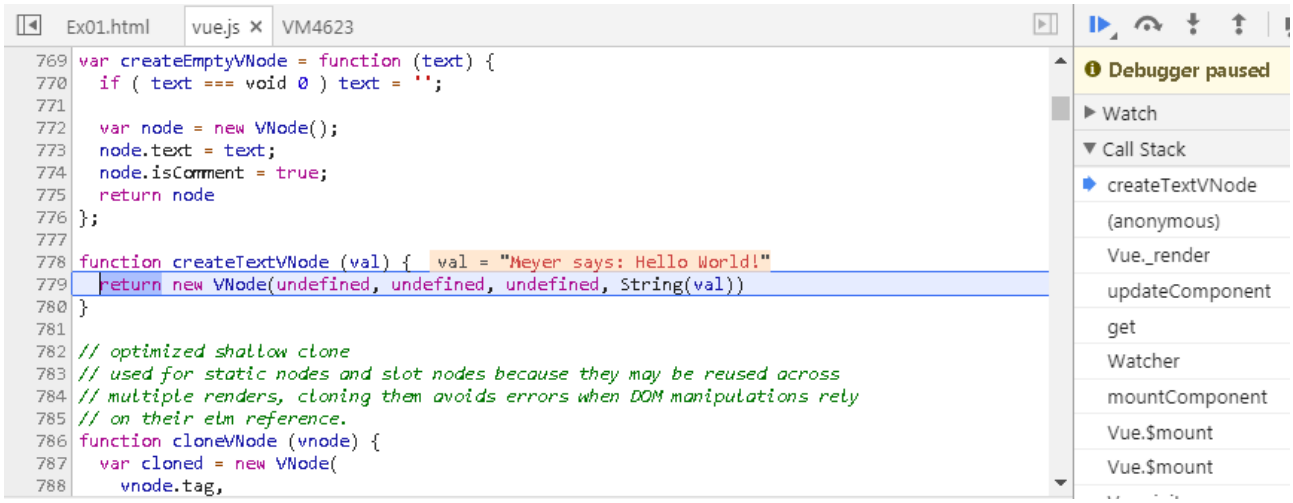


93-j. A função "***_s(...)***" invoca a função ***toString***:

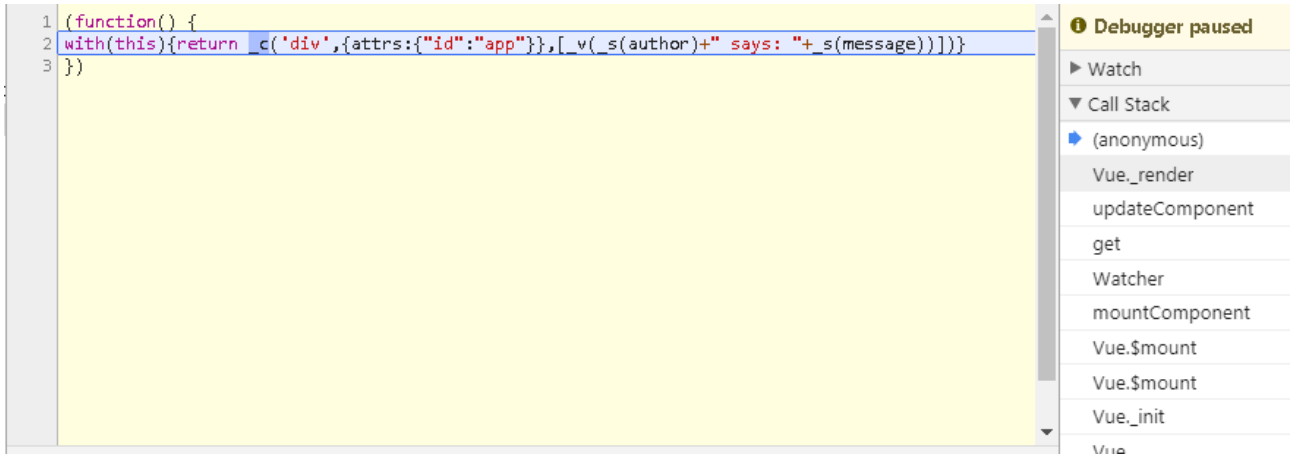


93-k. Sobre o resultado da concatenação das duas variáveis (author e message), é invocada a função Vue "***_v(...)***":

93-l. Esta, por sua vez, chama ***createTextVNode***:

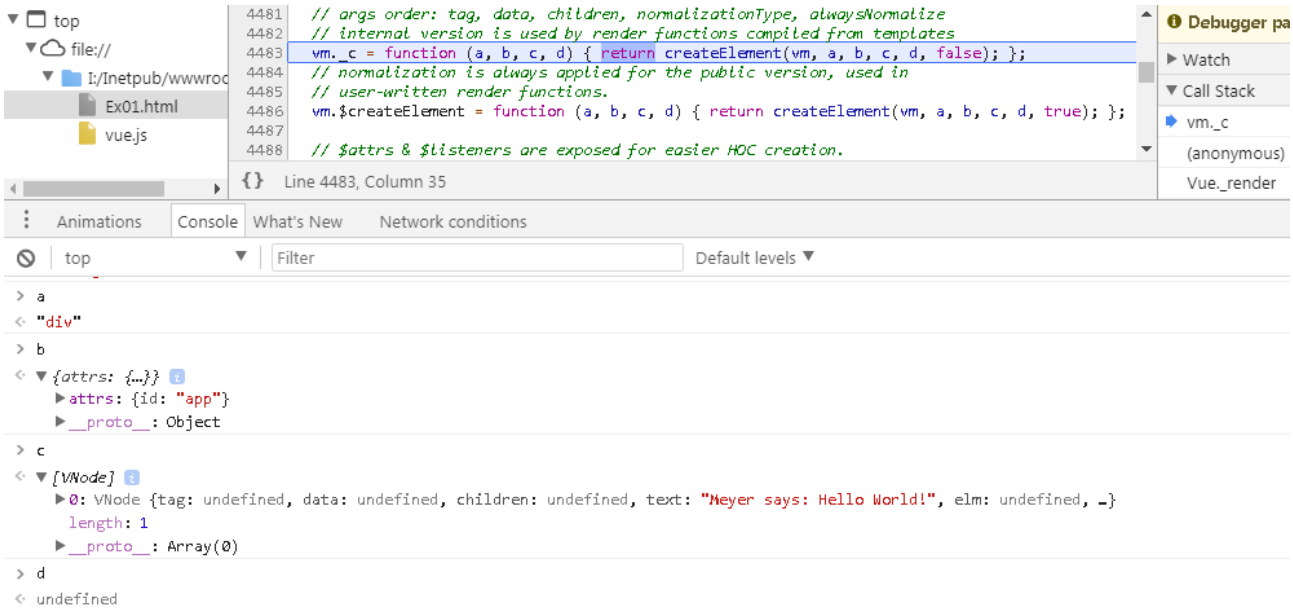


93-m. Resolvida a concatenação das strings, é chamada a função "***_c(...)***", com uma série de argumentos, incluindo os atributos a serem incluídos:

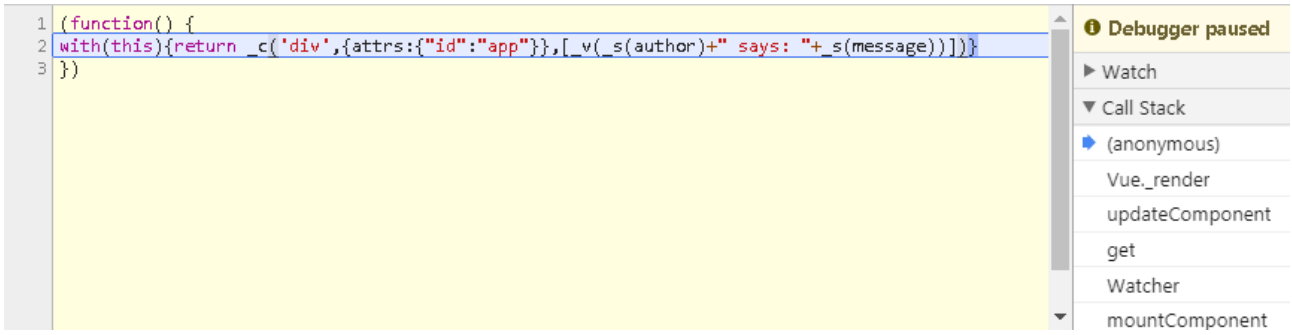


VUE – Substituição de variáveis

93-n. A função "***_c(...)***" chama ***createElement***, como vemos explicitamente:

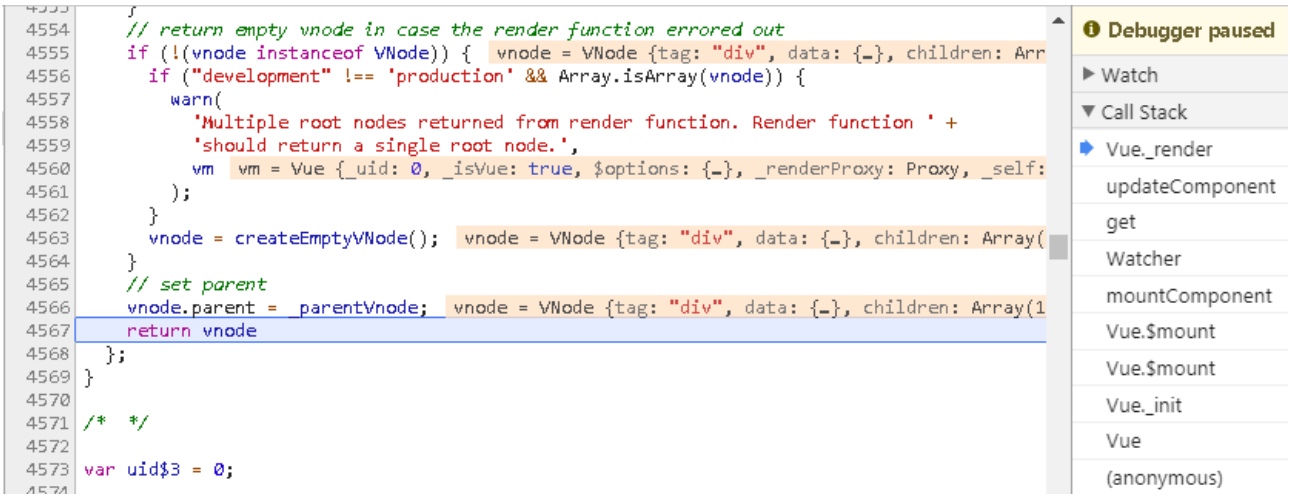


93-o. O retorno se dá para a conclusão da função de renderização:

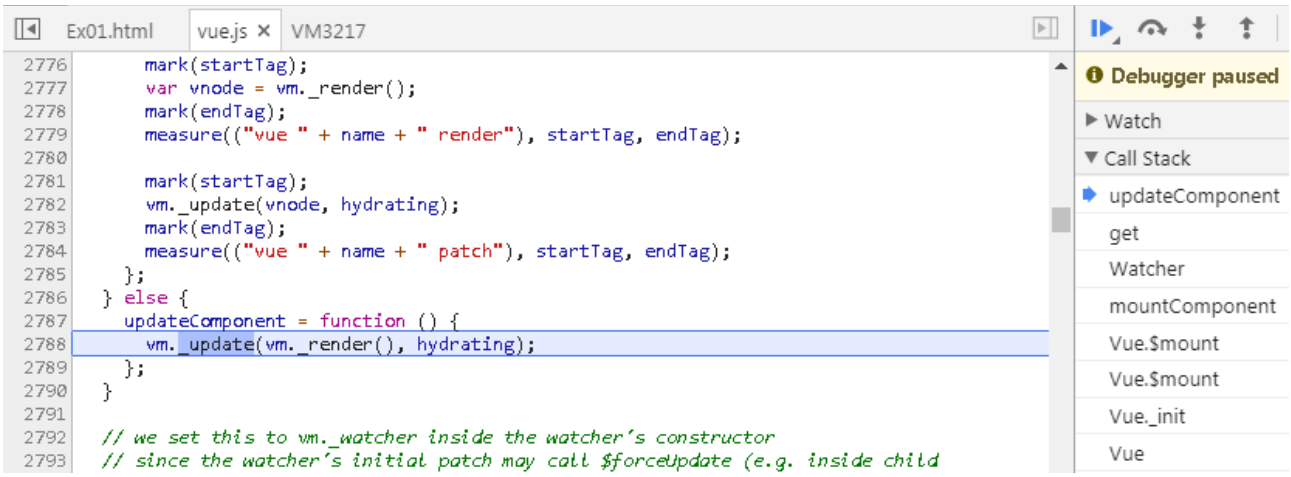


VUE – Substituição de variáveis

94. O processo de renderização retorna um “**vnode**”:

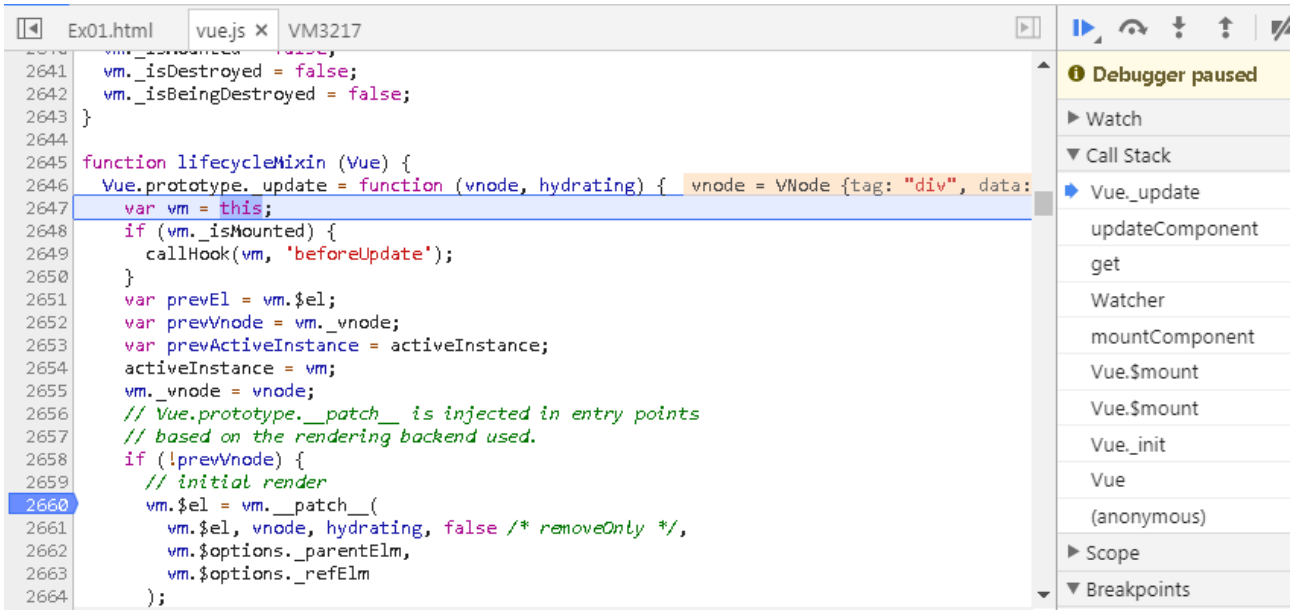


95. O processamento retorna à função ***updateComponent***:

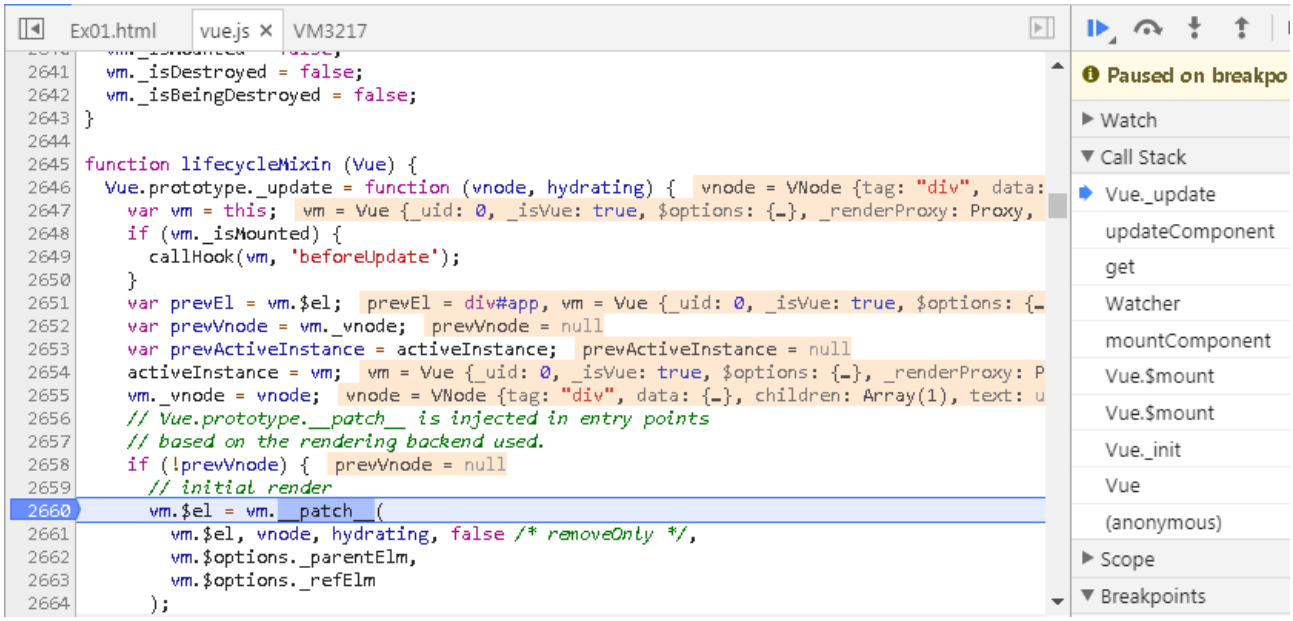


VUE – Substituição de variáveis

96. Terminada a chamada à ***vm._render***, o processamento é desviado para ***vm._update***:

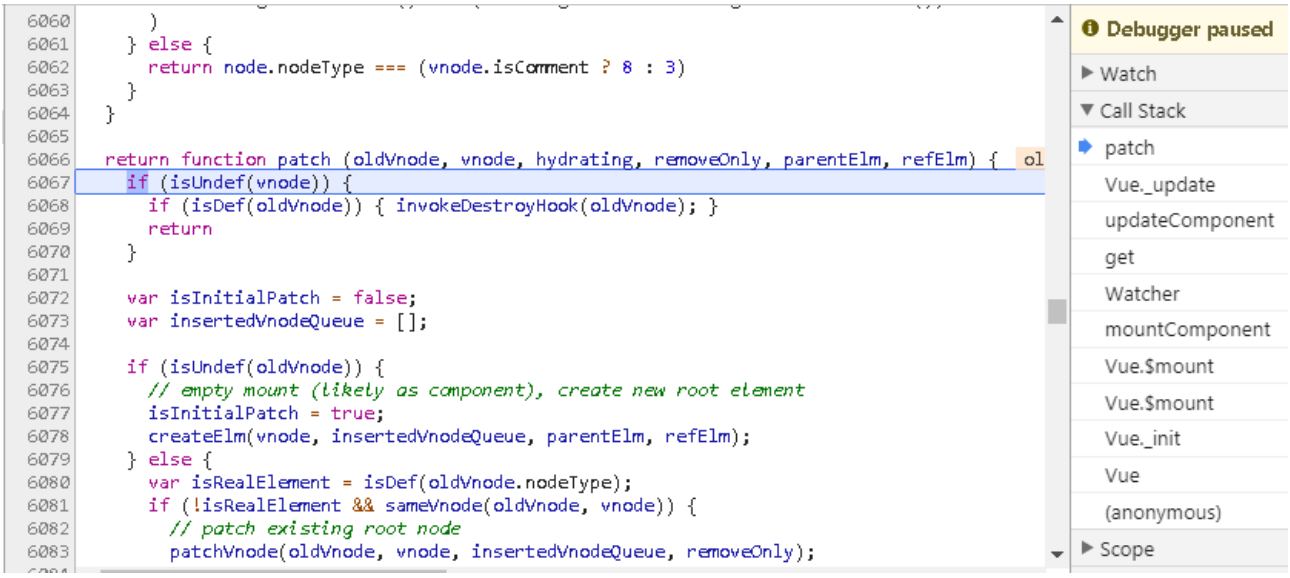


97. Na sequência de processamento, é chamada a função ***vm.__patch__***:



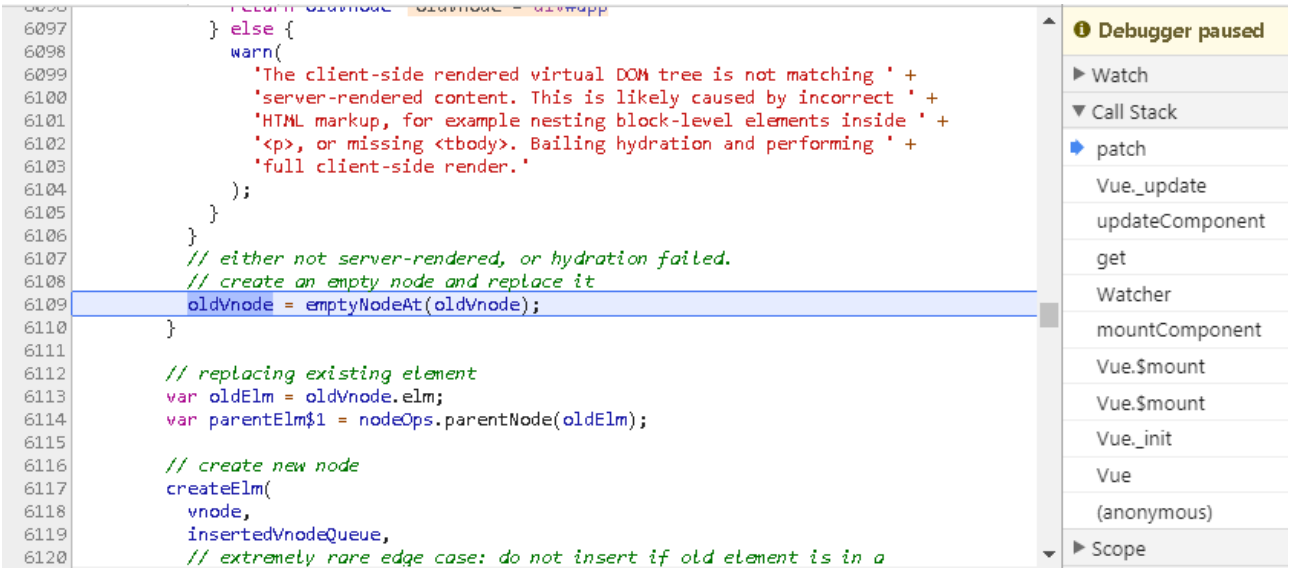
VUE – Substituição de variáveis

98. A função ***__patch__*** gerencia a colocação do novo nodo. Vejamos como ela vai tratar este nodo texto gerado pela função de renderização:




```
6060        )
6061      } else {
6062        return node.nodeType === (vnode.isComment ? 8 : 3)
6063      }
6064    }
6065
6066    return function patch (oldVnode, vnode, hydrating, removeOnly, parentElm, refElm) {  ol
6067      if (isUndef(vnode)) {
6068        if (isDef(oldVnode)) { invokeDestroyHook(oldVnode); }
6069        return
6070      }
6071
6072      var isInitialPatch = false;
6073      var insertedVnodeQueue = [];
6074
6075      if (isUndef(oldVnode)) {
6076        // empty mount (likely as component), create new root element
6077        isInitialPatch = true;
6078        createElm(vnode, insertedVnodeQueue, parentElm, refElm);
6079      } else {
6080        var isRealElement = isDef(oldVnode.nodeType);
6081        if (!isRealElement && sameVnode(oldVnode, vnode)) {
6082          // patch existing root node
6083          patchVnode(oldVnode, vnode, insertedVnodeQueue, removeOnly);
```

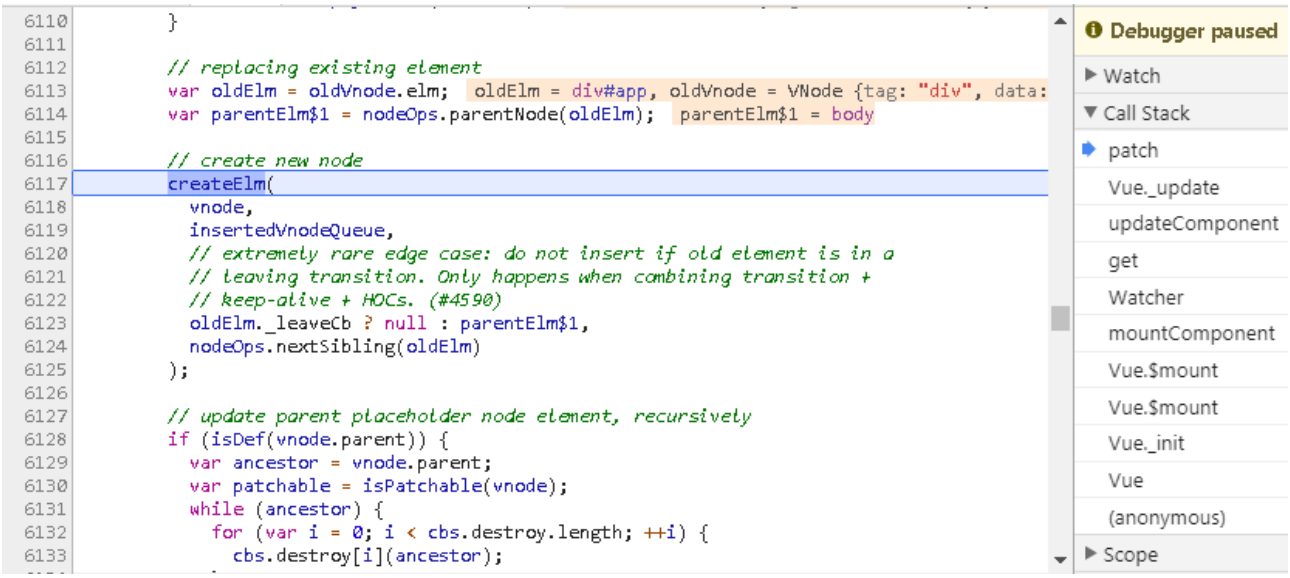
99. Guarda o ponteiro para o velho nodo:



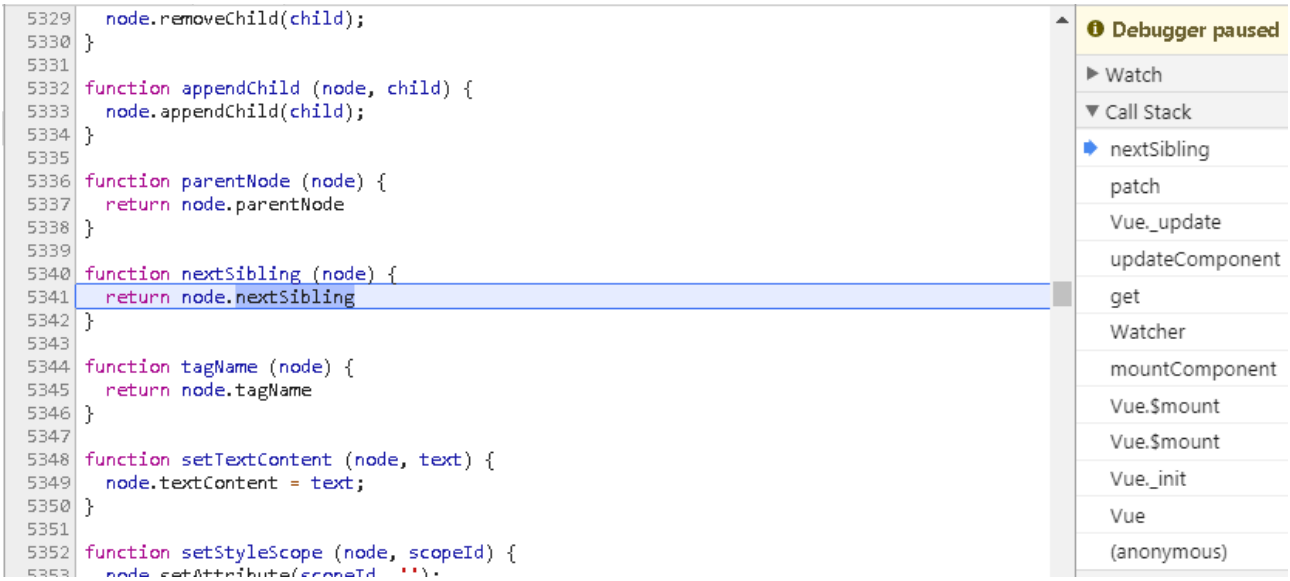

```
6097          } else {
6098            warn(
6099              'The client-side rendered virtual DOM tree is not matching ' +
6100              'server-rendered content. This is likely caused by incorrect ' +
6101              'HTML markup, for example nesting block-level elements inside ' +
6102              '<p>, or missing <tbody>. Bailing hydration and performing ' +
6103              'full client-side render.'
6104            );
6105          }
6106        }
6107        // either not server-rendered, or hydration failed.
6108        // create an empty node and replace it
6109        oldVnode = emptyNodeAt(oldVnode);
6110      }
6111
6112      // replacing existing element
6113      var oldElm = oldVnode.elm;
6114      var parentElm$1 = nodeOps.parentNode(oldElm);
6115
6116      // create new node
6117      createElm(
6118        vnode,
6119        insertedVnodeQueue,
6120        // extremely rare edge case: do not insert if old element is in a
```

VUE – Substituição de variáveis

100. Para criar o novo nodo, chama a função ***createElm***:

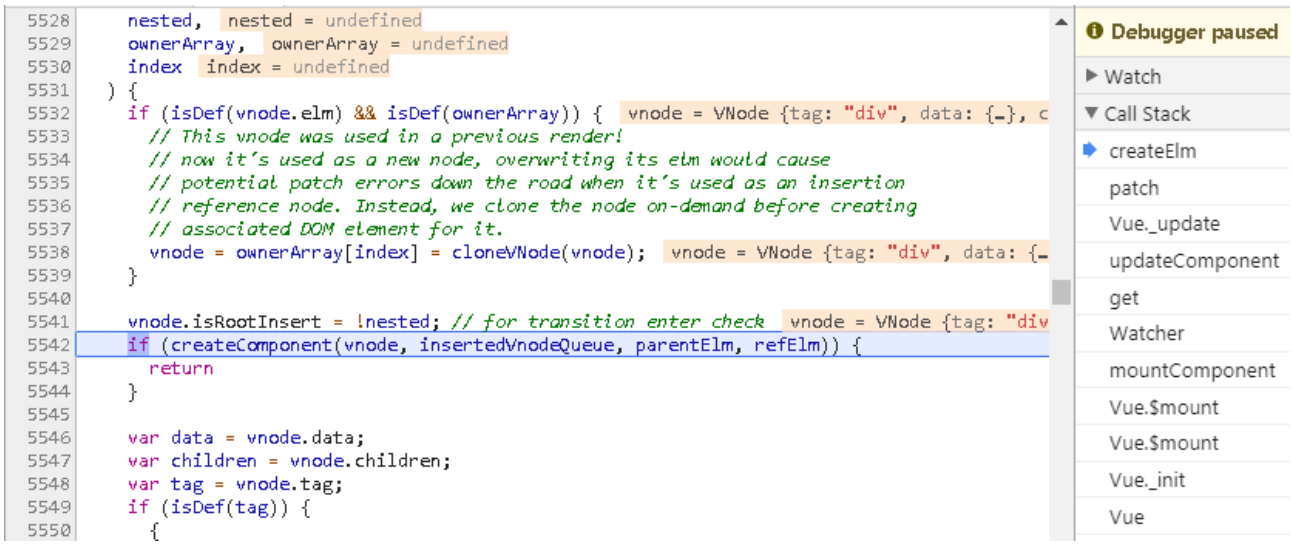


101. Na resolução dos parâmetros, é chamada a função ***nextSibling*** sobre o nodo anterior (**oldElm**). Este método dos nós do DOM retorna o próximo nó filho do pai especificado pelo nó referência.

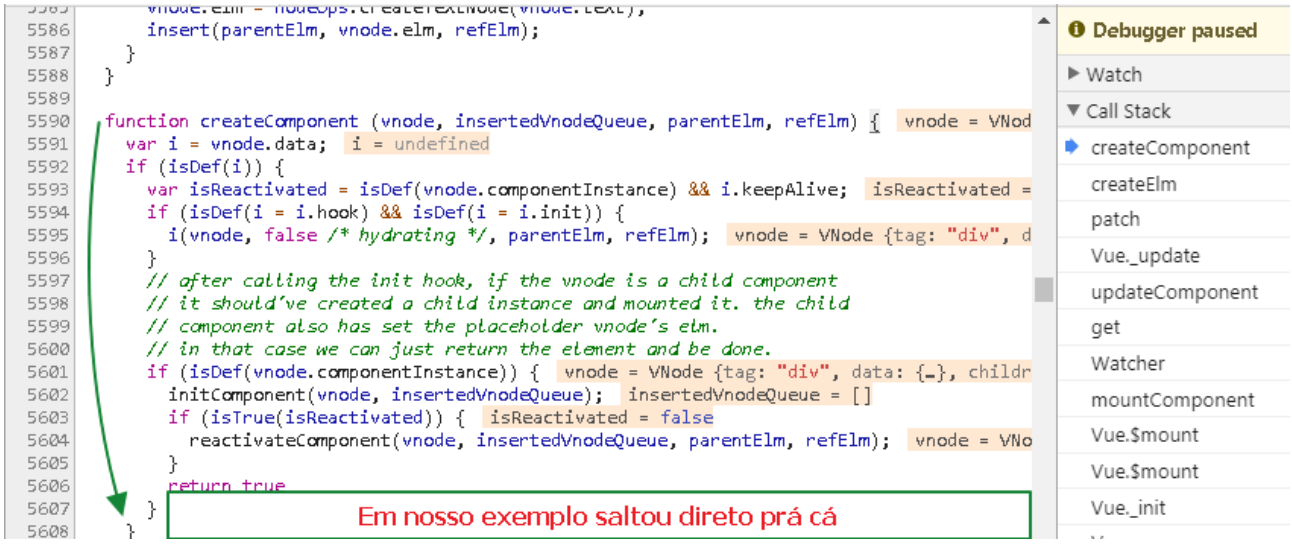


VUE – Substituição de variáveis

102. É feito um teste através da função ***createComponent***:

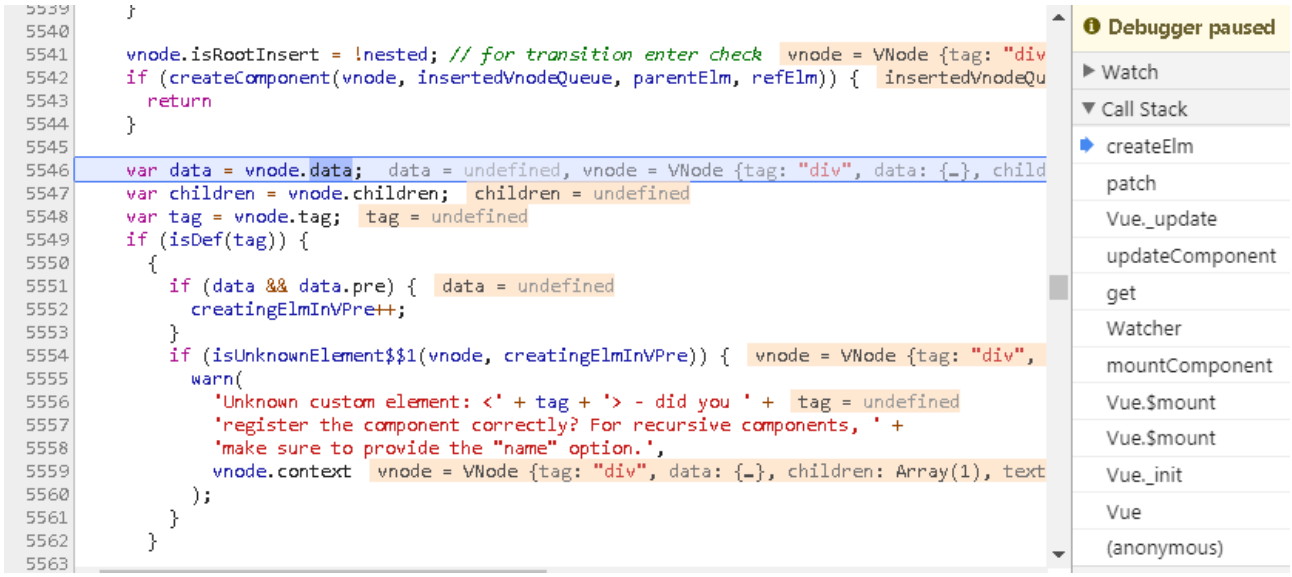


103. Como o nó referenciado está indefinido, o processamento salta para o fim da rotina:

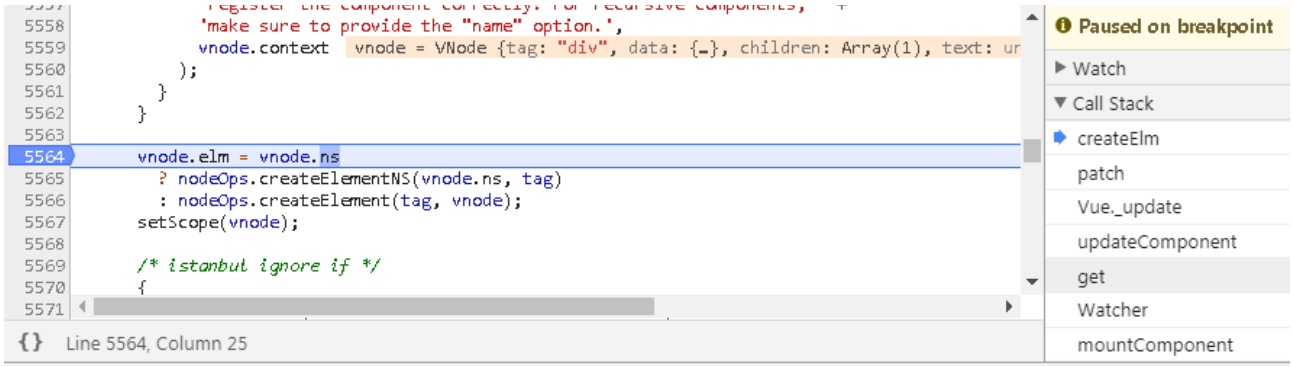


VUE – Substituição de variáveis

104. Algumas providências ...



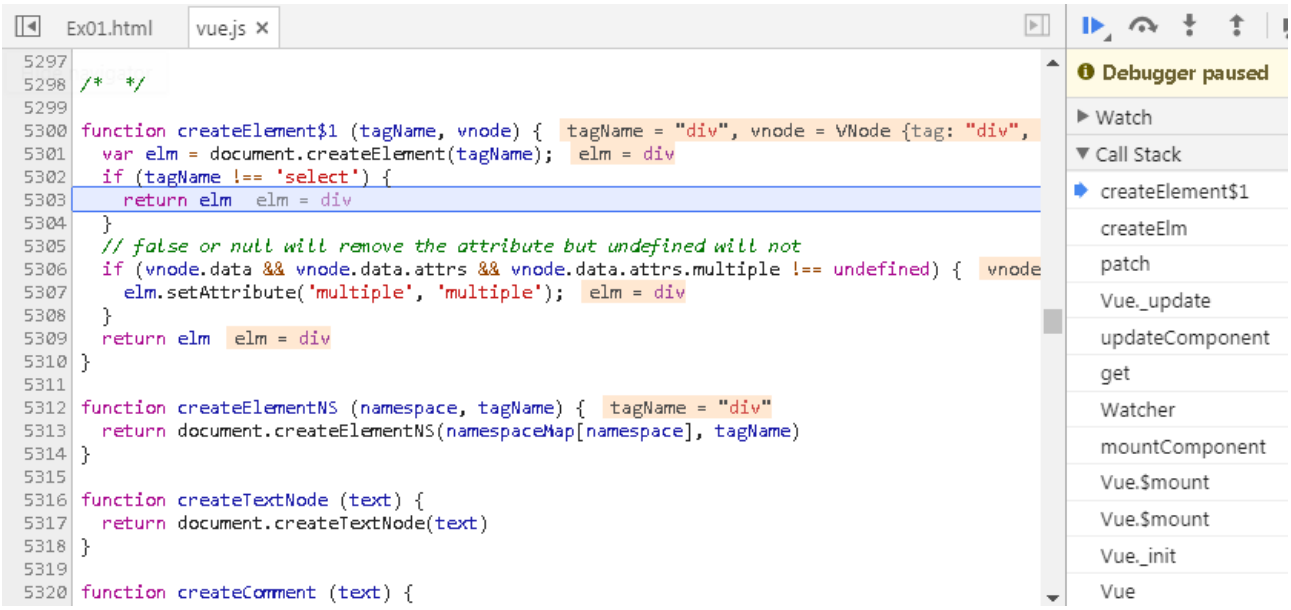
104-a. Na linha 5564 a chamada a ***createElementNS*** invoca a função ***createElement$1***:



VUE – Substituição de variáveis

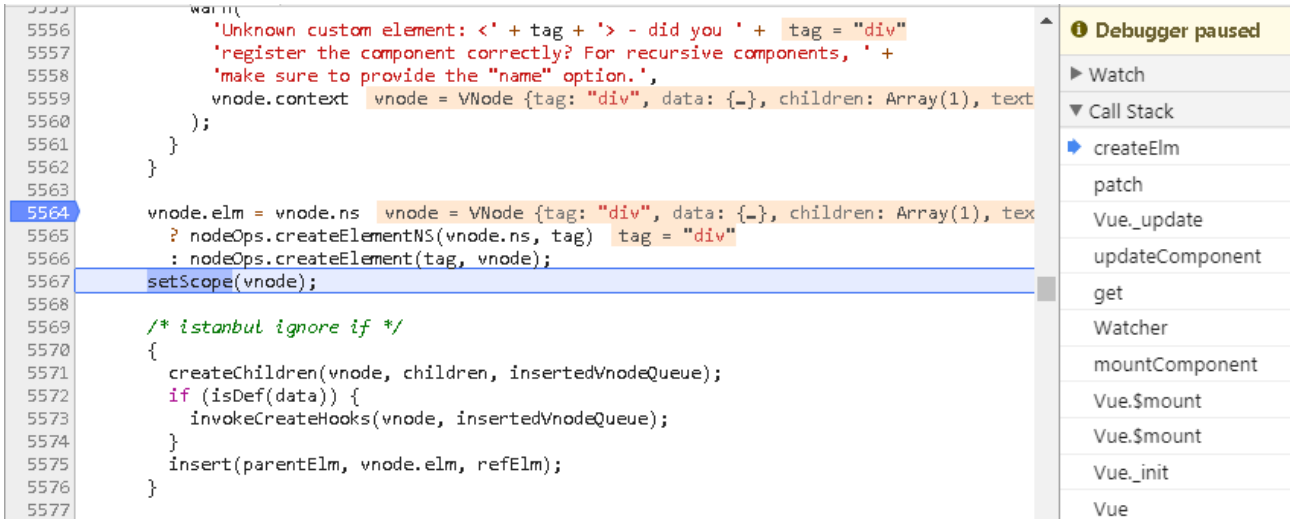
105. E createElement$1 cria um HTLMElement do tipo determinado pelo parâmetro tagName, que em nosso caso é “DIV”:

106. O produto retornado é HTMLElement do tipo container DIV:

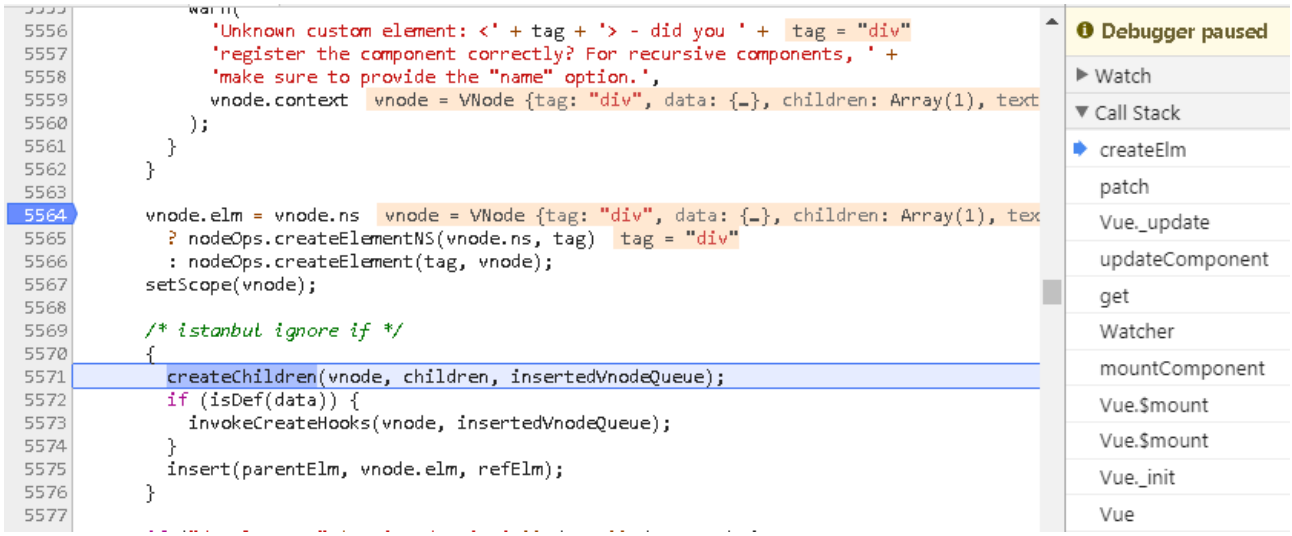


VUE – Substituição de variáveis

107. Retorno à createElm:

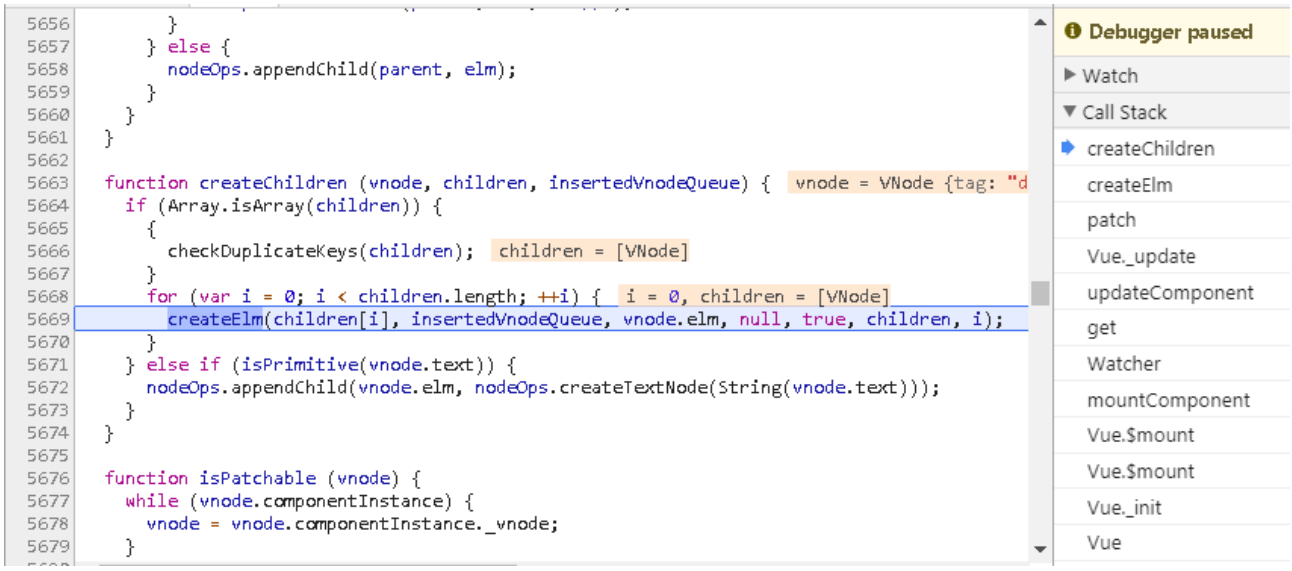


108. Em seguida serão criados os nós filhos:

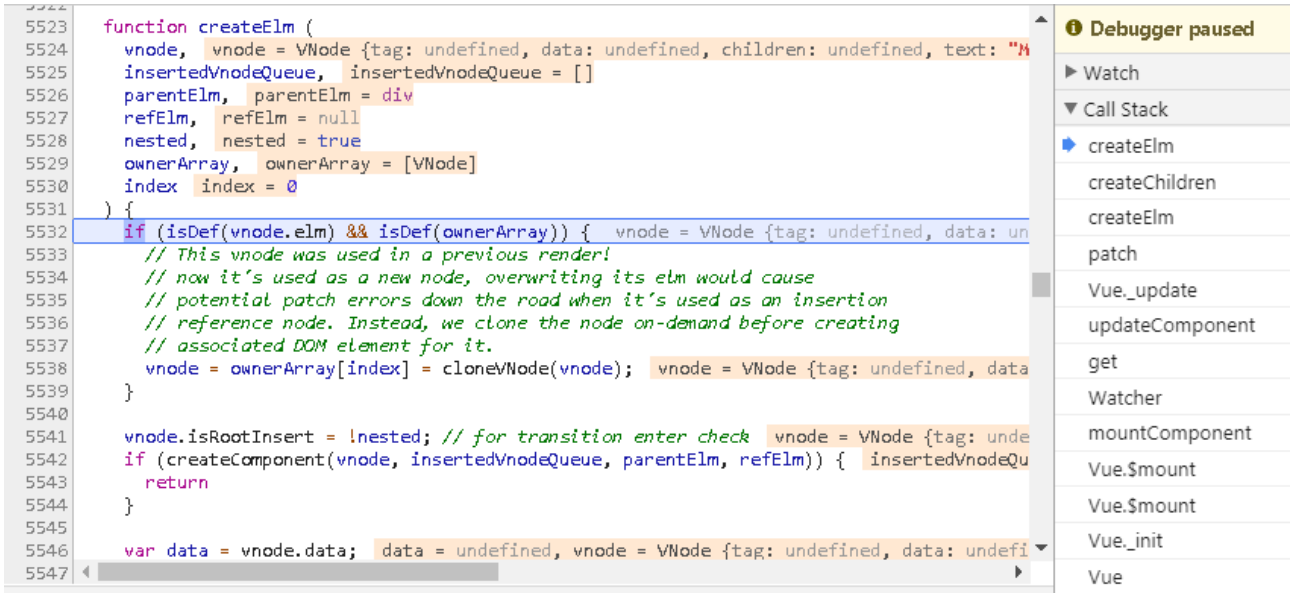


VUE – Substituição de variáveis

109. Aqui detalhamos a função ***createChildren.*** Para cada nó filho contido na estrutura de objeto vnode é criado um elemento correspondente, através de ***createElm***:



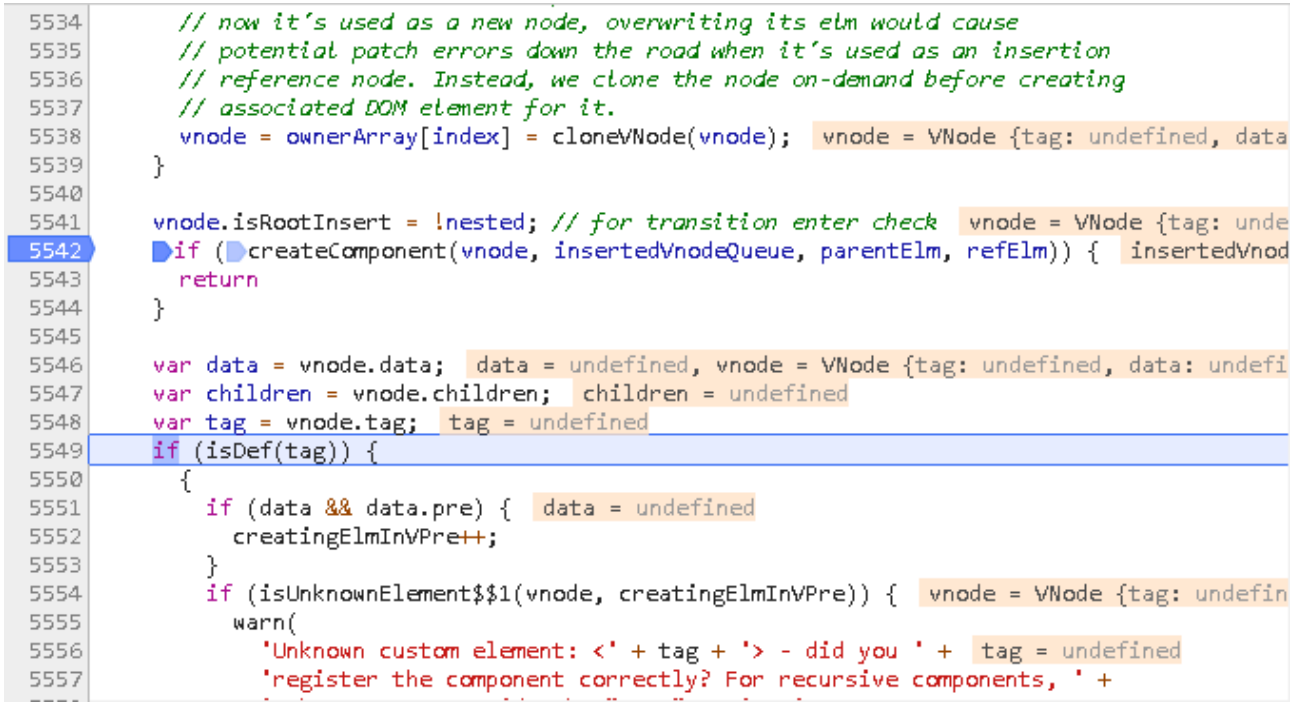
110. Esta função é chamada. Vamos concentrar nossa atenção na linha 5542:



VUE – Substituição de variáveis

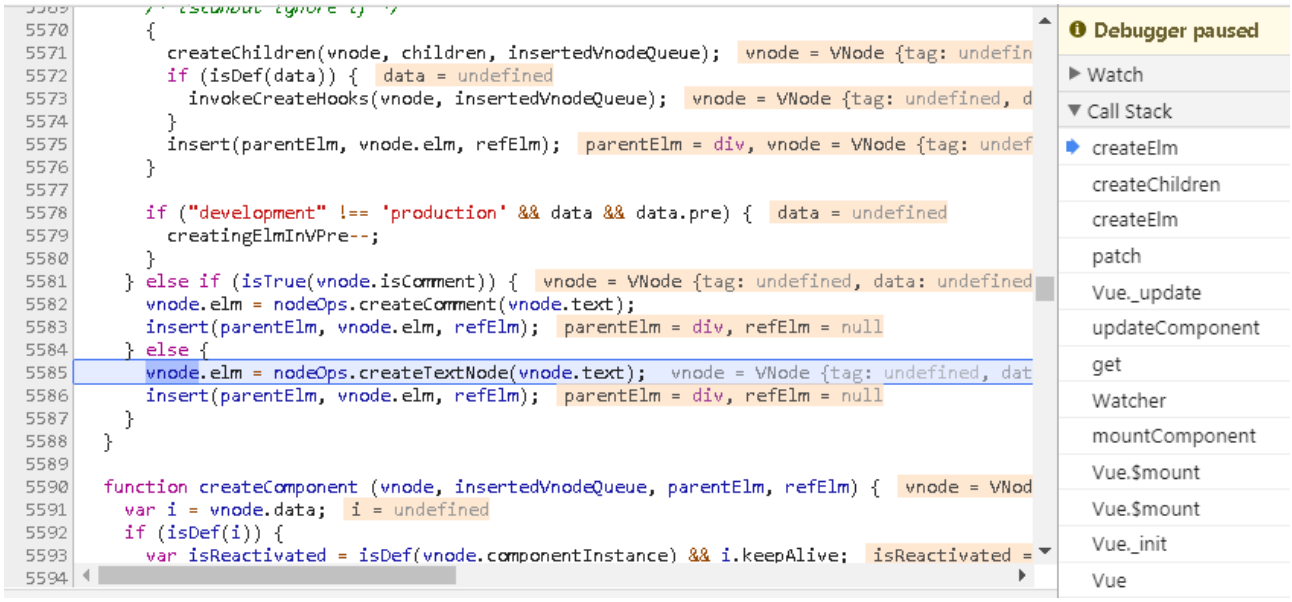
111. É chamada a função ***createComponent*** pois, como dissemos anteriormente, quando a template não é explícita em termos de tags de usuário, a template usada é aquela declarada pelo parâmetro "el" das opções declaradas na chamada de instanciação Vue.

112. A função retorna à ***createElm***:



VUE – Substituição de variáveis

113. E, na linha 5585, é chamada a função ***createTextNode***:



114. E esta chama o método de mesmo nome do objeto **document** do DOM:

VUE – Substituição de variáveis

115. No retorno de ***createTextNode*** é chamada a função ***insert***:

```
5574        }
5575        insert(parentElm, vnode.elm, refElm);  parentElm = div, vnode = VNode {tag: undef
5576      }
5577
5578      if ("development" !== 'production' && data && data.pre) {  data = undefined
5579        creatingElmInVPre--;
5580      }
5581    } else if (isTrue(vnode.isComment)) {  vnode = VNode {tag: undefined, data: undefined
5582      vnode.elm = nodeOps.createComment(vnode.text);
5583      insert(parentElm, vnode.elm, refElm);  parentElm = div, refElm = null
5584    } else {
5585      vnode.elm = nodeOps.createTextNode(vnode.text);  vnode = VNode {tag: undefined, dat
5586      insert(parentElm, vnode.elm, refElm);
5587    }
5588  }
5589
5590  function createComponent (vnode, insertedVnodeQueue, parentElm, refElm) {
5591    var i = vnode.data;
5592    if (isDef(i)) {
5593      var isReactivated = isDef(vnode.componentInstance) && i.keepAlive;
5594      if (isDef(i = i.hook) && isDef(i = i.init)) {
5595        i(vnode, false /* hydrating */, parentElm, refElm);
5596      }
5597      // after calling the init hook, if the vnode is a child component
```

116. A função ***Insert*** chama o método ***appendchild*** de **nodeOps**:

```
5644      }
5645    }
5646    // unlike a newly created component,
5647    // a reactivated keep-alive component doesn't insert itself
5648    insert(parentElm, vnode.elm, refElm);
5649  }
5650
5651  function insert (parent, elm, ref$$1) {  parent = div, elm = text, ref$$1 = null
5652    if (isDef(parent)) {
5653      if (isDef(ref$$1)) {  ref$$1 = null
5654        if (ref$$1.parentNode === parent) {  parent = div
5655          nodeOps.insertBefore(parent, elm, ref$$1);  elm = text
5656        }
5657      } else {
5658        nodeOps.appendChild(parent, elm);
5659      }
5660    }
5661  }
5662
5663  function createChildren (vnode, children, insertedVnodeQueue) {
5664    if (Array.isArray(children)) {
5665      {
5666        checkDuplicateKeys(children);
5667      }
5668      for (var i = 0; i < children.length; ++i) {
```

VUE – Substituição de variáveis

117. O método ***appendchild*** de **nodeOps** chama o método ***appendchild*** de **node**:

```
5323
5324 function insertBefore (parentNode, newNode, referenceNode) {
5325   parentNode.insertBefore(newNode, referenceNode);
5326 }
5327
5328 function removeChild (node, child) {
5329   node.removeChild(child);
5330 }
5331
5332 function appendChild (node, child) {   node = div, child = text
5333   node.appendChild(child);
5334 }
5335
5336 function parentNode (node) {   node = div
5337   return node.parentNode
5338 }
5339
5340 function nextSibling (node) {   node = div
5341   return node.nextSibling
5342 }
5343
5344 function tagName (node) {   node = div
5345   return node.tagName
5346 }
5347
```

118. Retornamos à ***insert***:

```
5649   }
5650
5651   function insert (parent, elm, ref$$1) {   parent = div, elm = text, ref$$1 = null
5652     if (isDef(parent)) {
5653       if (isDef(ref$$1)) {   ref$$1 = null
5654         if (ref$$1.parentNode === parent) {   parent = div
5655           nodeOps.insertBefore(parent, elm, ref$$1);   elm = text
5656         }
5657       } else {
5658         nodeOps.appendChild(parent, elm);   parent = div, elm = text
5659       }
5660     }
5661   }
5662
5663   function createChildren (vnode, children, insertedVnodeQueue) {
5664     if (Array.isArray(children)) {
5665       {
5666         checkDuplicateKeys(children);
5667       }
5668       for (var i = 0; i < children.length; ++i) {
5669         createElm(children[i], insertedVnodeQueue, vnode.elm, null, true, children, i);
5670       }
5671     } else if (isPrimitive(vnode.text)) {
5672       nodeOps.appendChild(vnode.elm, nodeOps.createTextNode(String(vnode.text)));
5673     }
```

119. Retornamos à ***createElm***:

```
5576        }
5577
5578        if ("development" !== 'production' && data && data.pre) {  data = undefined
5579          creatingElmInVPre--;
5580        }
5581      } else if (isTrue(vnode.isComment)) {  vnode = VNode {tag: undefined, data: undefined
5582        vnode.elm = nodeOps.createComment(vnode.text);
5583        insert(parentElm, vnode.elm, refElm);  parentElm = div, refElm = null
5584      } else {
5585        vnode.elm = nodeOps.createTextNode(vnode.text);  vnode = VNode {tag: undefined, dat
5586        insert(parentElm, vnode.elm, refElm);  parentElm = div, refElm = null
5587      }
5588    }
5589
5590    function createComponent (vnode, insertedVnodeQueue, parentElm, refElm) {  vnode = VNod
5591      var i = vnode.data;
5592      if (isDef(i)) {
5593        var isReactivated = isDef(vnode.componentInstance) && i.keepAlive;  vnode = VNode {
5594        if (isDef(i = i.hook) && isDef(i = i.init)) {
5595          i(vnode, false /* hydrating */, parentElm, refElm);  vnode = VNode {tag: undefine
5596        }
5597        // after calling the init hook, if the vnode is a child component
5598        // it should've created a child instance and mounted it. the child
5599        // component also has set the placeholder vnode's elm.
```

120. Retornamos à ***createChildren***:

```
5657        } else {
5658          nodeOps.appendChild(parent, elm);
5659        }
5660      }
5661    }
5662
5663    function createChildren (vnode, children, insertedVnodeQueue) {  vnode = VNode {tag: "d
5664      if (Array.isArray(children)) {
5665        {
5666          checkDuplicateKeys(children);  children = [VNode]
5667        }
5668        for (var i = 0; i < children.length; ++i) {  i = 1, children = [VNode]
5669          createElm(children[i], insertedVnodeQueue, vnode.elm, null, true, children, i);
5670        }
5671      } else if (isPrimitive(vnode.text)) {  vnode = VNode {tag: "div", data: {…}, children
5672        nodeOps.appendChild(vnode.elm, nodeOps.createTextNode(String(vnode.text)));
5673      }
5674    }
5675
5676    function isPatchable (vnode) {
5677      while (vnode.componentInstance) {
5678        vnode = vnode.componentInstance._vnode;
5679      }
5680      return isDef(vnode.tag)
```

121. Em seguida o Vue chama invokeCreateHooks.

```
5560            );
5561          }
5562        }
5563
5564        vnode.elm = vnode.ns  vnode = VNode {tag: "div", data: {…}, children: Array(1), tex
5565          ? nodeOps.createElementNS(vnode.ns, tag)  tag = "div"
5566          : nodeOps.createElement(tag, vnode);
5567        setScope(vnode);
5568
5569        /* istanbul ignore if */
5570        {
5571          createChildren(vnode, children, insertedVnodeQueue);  vnode = VNode {tag: "div",
5572          if (isDef(data)) {  data = {attrs: {…}}
5573            invokeCreateHooks(vnode, insertedVnodeQueue);  vnode = VNode {tag: "div", data:
5574          }
5575          insert(parentElm, vnode.elm, refElm);  parentElm = body, vnode = VNode {tag: "div
5576        }
5577
5578        if ("development" !== 'production' && data && data.pre) {  data = {attrs: {…}}
5579          creatingElmInVPre--;
5580        }
5581      } else if (isTrue(vnode.isComment)) {  vnode = VNode {tag: "div", data: {…}, children
5582        vnode.elm = nodeOps.createComment(vnode.text);
5583        insert(parentElm, vnode.elm, refElm);  parentElm = body, refElm = text
```

122. As **hooks** do Vue são seus eventos, a saber, ***activate***, ***create***, ***destroy***, ***remove*** e ***update***. Neste momento, os **hooks** da seção create são chamados um a um: ***updateAttrs***, ***updateClass***, ***updateDOMListeners***, ***updateDOMProps***, ***updateStyle***, ***_enter***, ***create*** e ***updateDirectives***. Os hooks do usuário (constantes do corpo de opções de chamada do Vue) podem se sobrepor aos defaults, caso sejam declarados.

```
5673    }
5674  }
5675
5676  function isPatchable (vnode) {
5677    while (vnode.componentInstance) {
5678      vnode = vnode.componentInstance._vnode;
5679    }
5680    return isDef(vnode.tag)
5681  }
5682
5683  function invokeCreateHooks (vnode, insertedVnodeQueue) {  vnode = VNode {tag: "div", da
5684    for (var i$1 = 0; i$1 < cbs.create.length; ++i$1) {  i$1 = 0
5685      cbs.create[i$1](emptyNode, vnode);
5686    }
5687    i = vnode.data.hook; // Reuse variable
5688    if (isDef(i)) {
5689      if (isDef(i.create)) { i.create(emptyNode, vnode); }
5690      if (isDef(i.insert)) { insertedVnodeQueue.push(vnode); }
5691    }
5692  }
5693
5694  // set scope id attribute for scoped CSS.
5695  // this is implemented as a special case to avoid the overhead
5696  // of going through the normal attribute patching process.
```

VUE – Substituição de variáveis

123. A hook de updateAttrs percorre cada um dos atributos do Vnode:

```
6292 /*  */
6293
6294 function updateAttrs (oldVnode, vnode) {  oldVnode = VNode {tag: "", data: {…}, children:
6295   var opts = vnode.componentOptions;  opts = undefined
6296   if (isDef(opts) && opts.Ctor.options.inheritAttrs === false) {
6297     return
6298   }
6299   if (isUndef(oldVnode.data.attrs) && isUndef(vnode.data.attrs)) {  oldVnode = VNode {tag
6300     return
6301   }
6302   var key, cur, old;  key = undefined, cur = undefined, old = undefined
6303   var elm = vnode.elm;  elm = div, vnode = VNode {tag: "div", data: {…}, children: Array(
6304   var oldAttrs = oldVnode.data.attrs || {};  oldAttrs = {}, oldVnode = VNode {tag: "", da
6305   var attrs = vnode.data.attrs || {};  attrs = {id: "app"}, vnode = VNode {tag: "div", da
6306   // clone observed objects, as the user probably wants to mutate it
6307   if (isDef(attrs.__ob__)) {  attrs = {id: "app"}
6308     attrs = vnode.data.attrs = extend({}, attrs);  vnode = VNode {tag: "div", data: {…},
6309   }
6310
6311   for (key in attrs) {
6312     cur = attrs[key];
6313     old = oldAttrs[key];
6314     if (old !== cur) {
6315       setAttr(elm, key, cur);
```

124. E os ajusta por meio de ***setAttr***:

```
6303   var elm = vnode.elm;  elm = div, vnode = VNode {tag: "div", data: {…}, children: Array(
6304   var oldAttrs = oldVnode.data.attrs || {};  oldAttrs = {}, oldVnode = VNode {tag: "", da
6305   var attrs = vnode.data.attrs || {};  attrs = {id: "app"}, vnode = VNode {tag: "div", da
6306   // clone observed objects, as the user probably wants to mutate it
6307   if (isDef(attrs.__ob__)) {  attrs = {id: "app"}
6308     attrs = vnode.data.attrs = extend({}, attrs);  vnode = VNode {tag: "div", data: {…},
6309   }
6310
6311   for (key in attrs) {  key = "id", attrs = {id: "app"}
6312     cur = attrs[key];  cur = "app"
6313     old = oldAttrs[key];  old = undefined, oldAttrs = {}
6314     if (old !== cur) {  cur = "app"
6315       setAttr(elm, key, cur);
6316     }
6317   }
6318   // #4391: in IE9, setting type can reset value for input[type=radio]
6319   // #6666: IE/Edge forces progress value down to 1 before setting a max
6320   /* istanbul ignore if */
6321   if ((isIE || isEdge) && attrs.value !== oldAttrs.value) {
6322     setAttr(elm, 'value', attrs.value);
6323   }
6324   for (key in oldAttrs) {
6325     if (isUndef(attrs[key])) {
6326       if (isXlink(key)) {
```

VUE – Substituição de variáveis

125. O atributo é analisado profundamente, quanto à nomenclatura utilizada e quanto ao tipo.

```
6332    }
6333  }
6334
6335  function setAttr (el, key, value) {  el = div, key = "id", value = "app"
6336    if (el.tagName.indexOf('-') > -1) {
6337      baseSetAttr(el, key, value);  key = "id", value = "app"
6338    } else if (isBooleanAttr(key)) {
6339      // set attribute for blank value
6340      // e.g. <option disabled>Select one</option>
6341      if (isFalsyAttrValue(value)) {  value = "app"
6342        el.removeAttribute(key);  el = div, key = "id"
6343      } else {
6344        // technically allowfullscreen is a boolean attribute for <iframe>,
6345        // but Flash expects a value of "true" when used on <embed> tag
6346        value = key === 'allowfullscreen' && el.tagName === 'EMBED'  value = "app", key = "
6347          ? 'true'
6348          : key;  key = "id"
6349        el.setAttribute(key, value);  el = div, value = "app"
6350      }
6351    } else if (isEnumeratedAttr(key)) {
6352      el.setAttribute(key, isFalsyAttrValue(value) || value === 'false' ? 'false' : 'true')
6353    } else if (isXlink(key)) {
6354      if (isFalsyAttrValue(value)) {
6355        el.removeAttributeNS(xlinkNS, getXlinkProp(key));
```

126. Até chamar a função ***baseSetAttr***:

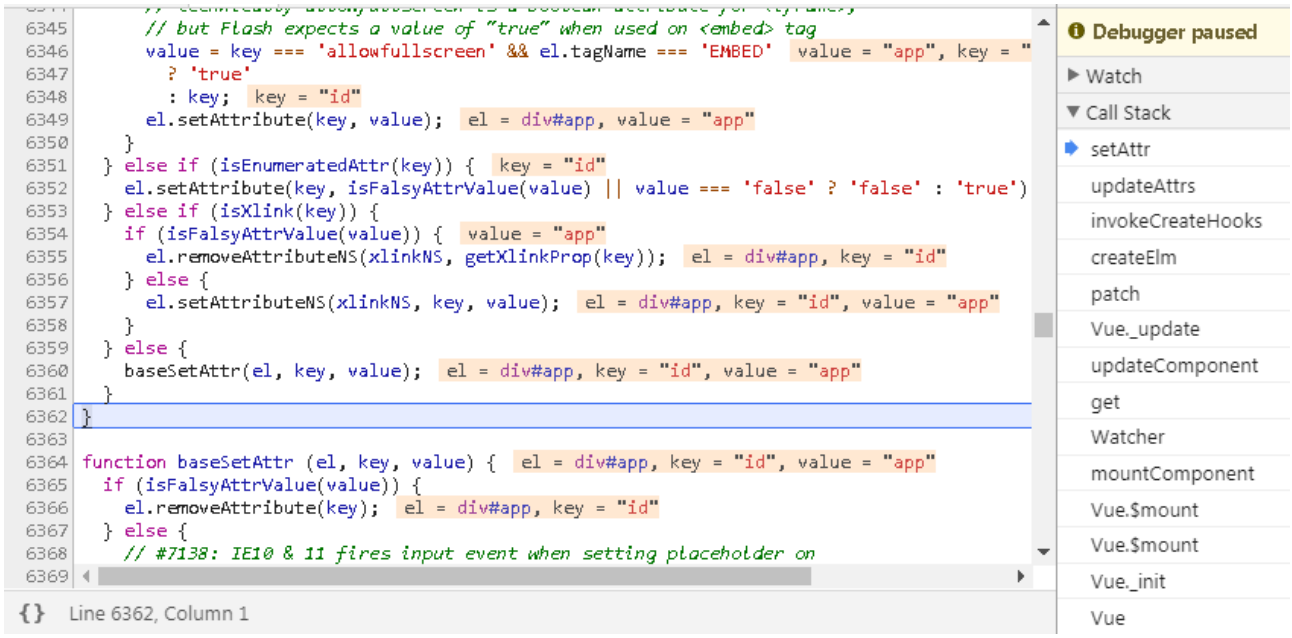
```
6345        // but Flash expects a value of "true" when used on <embed> tag
6346        value = key === 'allowfullscreen' && el.tagName === 'EMBED'  value = "app", key = "
6347          ? 'true'
6348          : key;  key = "id"
6349        el.setAttribute(key, value);  el = div, value = "app"
6350      }
6351    } else if (isEnumeratedAttr(key)) {  key = "id"
6352      el.setAttribute(key, isFalsyAttrValue(value) || value === 'false' ? 'false' : 'true')
6353    } else if (isXlink(key)) {
6354      if (isFalsyAttrValue(value)) {  value = "app"
6355        el.removeAttributeNS(xlinkNS, getXlinkProp(key));  el = div, key = "id"
6356      } else {
6357        el.setAttributeNS(xlinkNS, key, value);  el = div, key = "id", value = "app"
6358      }
6359    } else {
6360      baseSetAttr(el, key, value);
6361    }
6362  }
6363
6364  function baseSetAttr (el, key, value) {
6365    if (isFalsyAttrValue(value)) {
6366      el.removeAttribute(key);
6367    } else {
6368      // #7138: IE10 & 11 fires input event when setting placeholder on
```

VUE – Substituição de variáveis

127. O tratamento é feito levando-se em conta o tipo de tag que o elemento “el” se constitui.

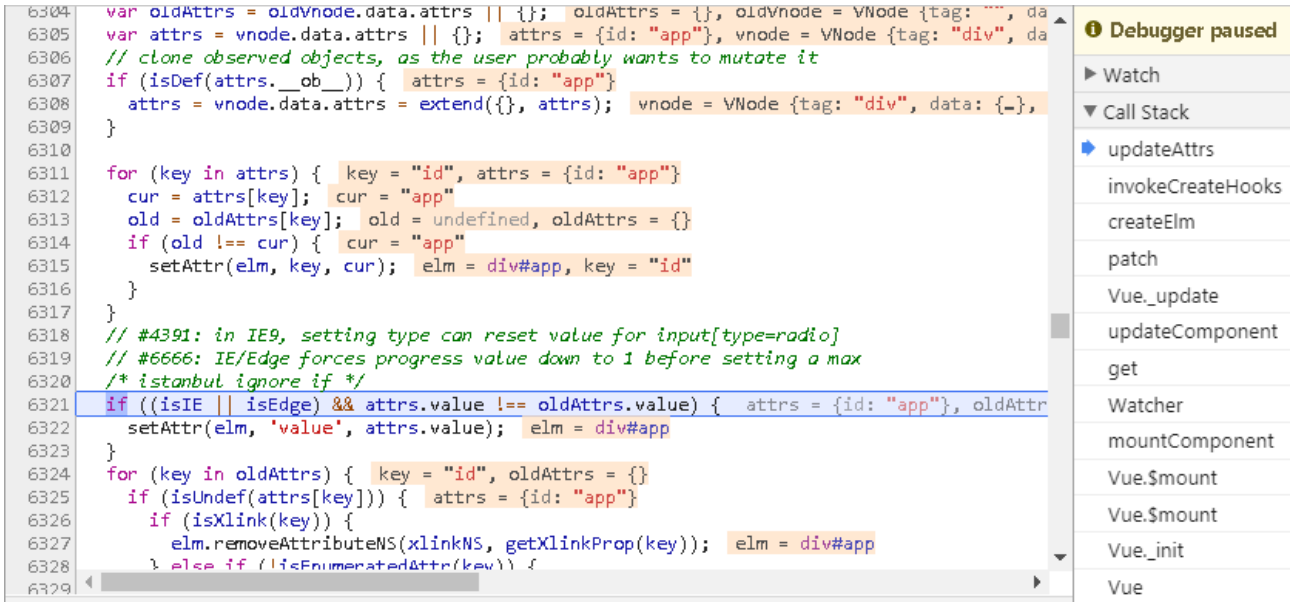
```
6364 function baseSetAttr (el, key, value) {  el = div#app, key = "id", value = "app"
6365   if (isFalsyAttrValue(value)) {
6366     el.removeAttribute(key);  el = div#app, key = "id"
6367   } else {
6368     // #7138: IE10 & 11 fires input event when setting placeholder on
6369     // <textarea>... block the first input event and remove the blocker
6370     // immediately.
6371     /* istanbul ignore if */
6372     if (
6373       isIE && !isIE9 &&
6374       el.tagName === 'TEXTAREA' &&  el = div#app
6375       key === 'placeholder' && !el.__ieph  key = "id"
6376     ) {
6377       var blocker = function (e) {  blocker = undefined
6378         e.stopImmediatePropagation();
6379         el.removeEventListener('input', blocker);  el = div#app, blocker = undefined
6380       };
6381       el.addEventListener('input', blocker);  el = div#app, blocker = undefined
6382       // $flow-disable-line
6383       el.__ieph = true; /* IE placeholder patched */  el = div#app
6384     }
6385     el.setAttribute(key, value);  el = div#app, key = "id", value = "app"
6386   }
6387 }
```

128. Retornamos à ***setAttr***:

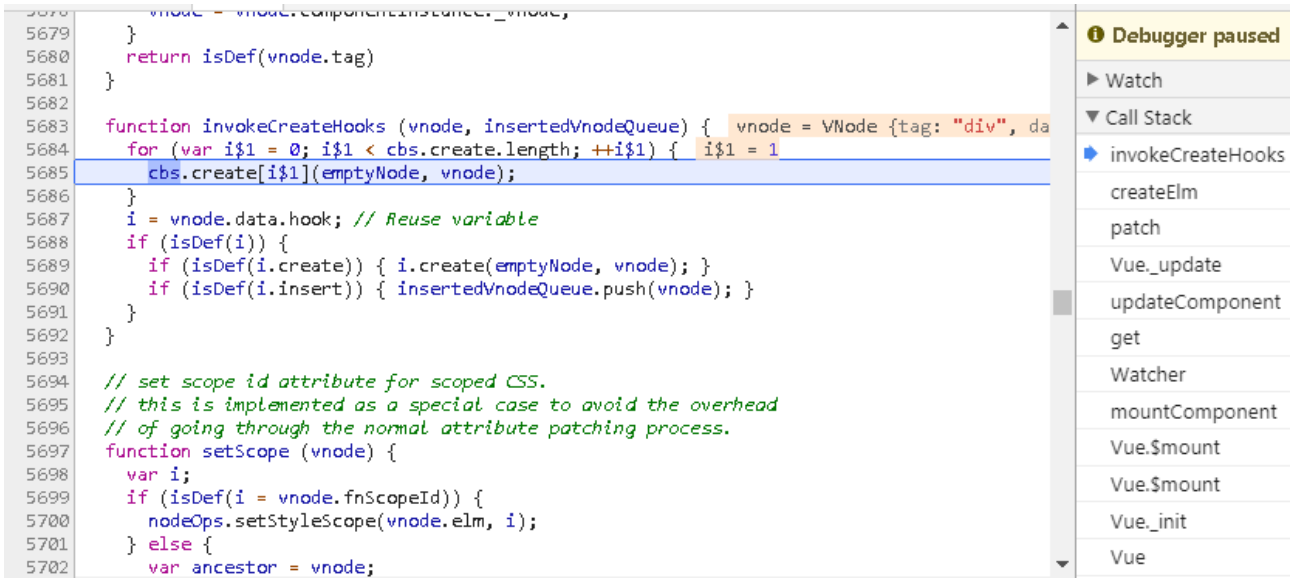


VUE – Substituição de variáveis

129. Retornamos à ***updateAttrs*** :



130. Retornamos à ***invokeCreateHooks*** :



VUE – Substituição de variáveis

131. Agora é chamado o hook de ***updateAttrs*** :

```
6391    update: updateAttrs
6392  }
6393
6394  /*  */
6395
6396  function updateClass (oldVnode, vnode) {  oldVnode = VNode {tag: "", data: {…}, children:
6397    var el = vnode.elm;  el = div#app
6398    var data = vnode.data;  data = {attrs: {…}}
6399    var oldData = oldVnode.data;  oldData = {}, oldVnode = VNode {tag: "", data: {…}, child
6400    if (
6401      isUndef(data.staticClass) &&  data = {attrs: {…}}
6402      isUndef(data.class) && (
6403        isUndef(oldData) || (  oldData = {}
6404          isUndef(oldData.staticClass) &&
6405          isUndef(oldData.class)
6406        )
6407      )
6408    ) {
6409      return
6410    }
6411
6412    var cls = genClassForVnode(vnode);
6413
6414    // handle transition classes
```

132. Retornamos a ***invokeCreateHooks*** :

```
5675
5676  function isPatchable (vnode) {
5677    while (vnode.componentInstance) {
5678      vnode = vnode.componentInstance._vnode;
5679    }
5680    return isDef(vnode.tag)
5681  }
5682
5683  function invokeCreateHooks (vnode, insertedVnodeQueue) {  vnode = VNode {tag: "div", da
5684    for (var i$1 = 0; i$1 < cbs.create.length; ++i$1) {  i$1 = 2
5685      cbs.create[i$1](emptyNode, vnode);
5686    }
5687    i = vnode.data.hook; // Reuse variable
5688    if (isDef(i)) {
5689      if (isDef(i.create)) { i.create(emptyNode, vnode); }
5690      if (isDef(i.insert)) { insertedVnodeQueue.push(vnode); }
5691    }
5692  }
5693
5694  // set scope id attribute for scoped CSS.
5695  // this is implemented as a special case to avoid the overhead
5696  // of going through the normal attribute patching process.
5697  function setScope (vnode) {
5698    var i;
```

Debugger paused

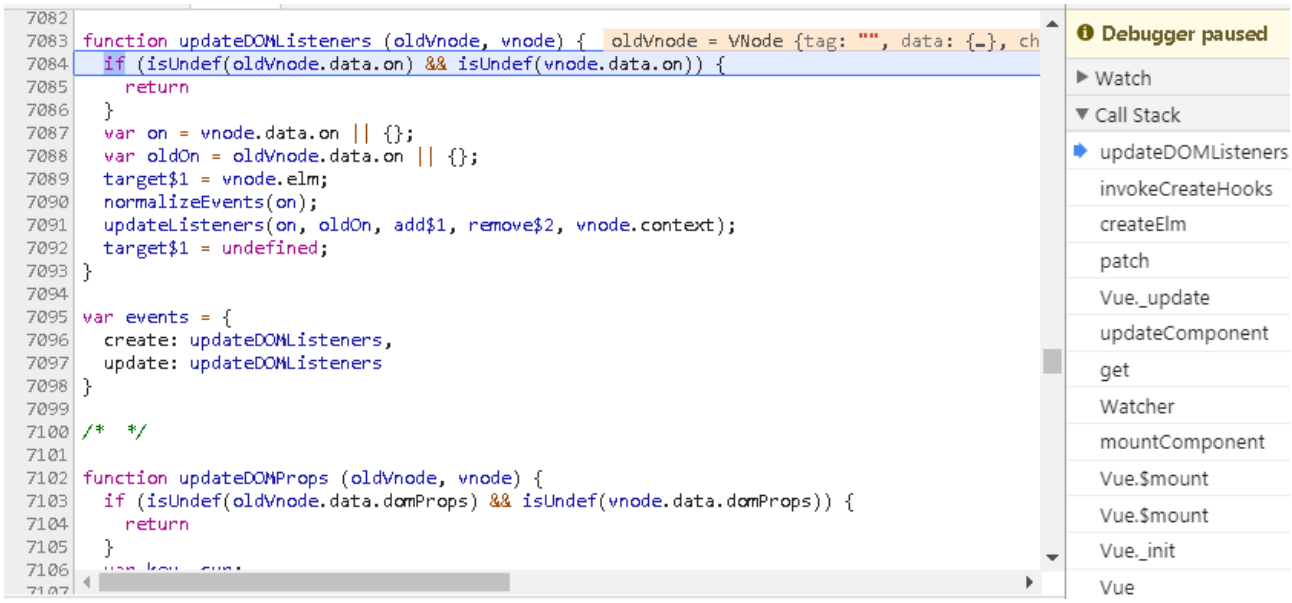
▶ Watch

▼ Call Stack

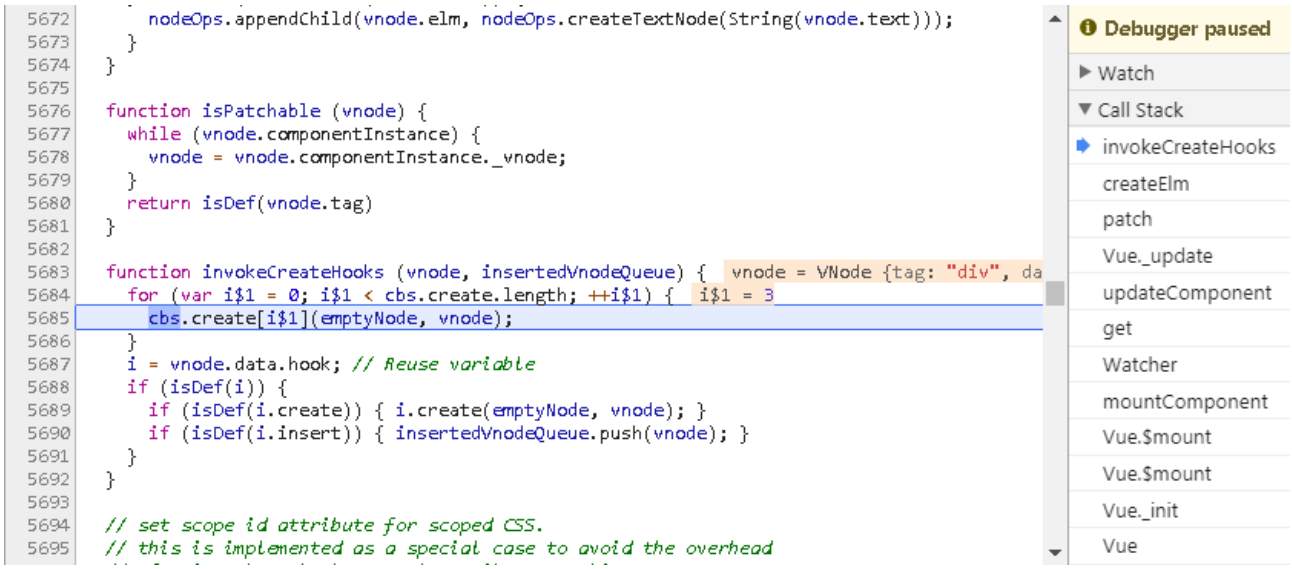
- invokeCreateHooks
- createElm
- patch
- Vue._update
- updateComponent
- get
- Watcher
- mountComponent
- Vue.$mount
- Vue.$mount
- Vue._init
- Vue

VUE – Substituição de variáveis

133. Agora é chamado o hook de ***updateDOMListeners*** :

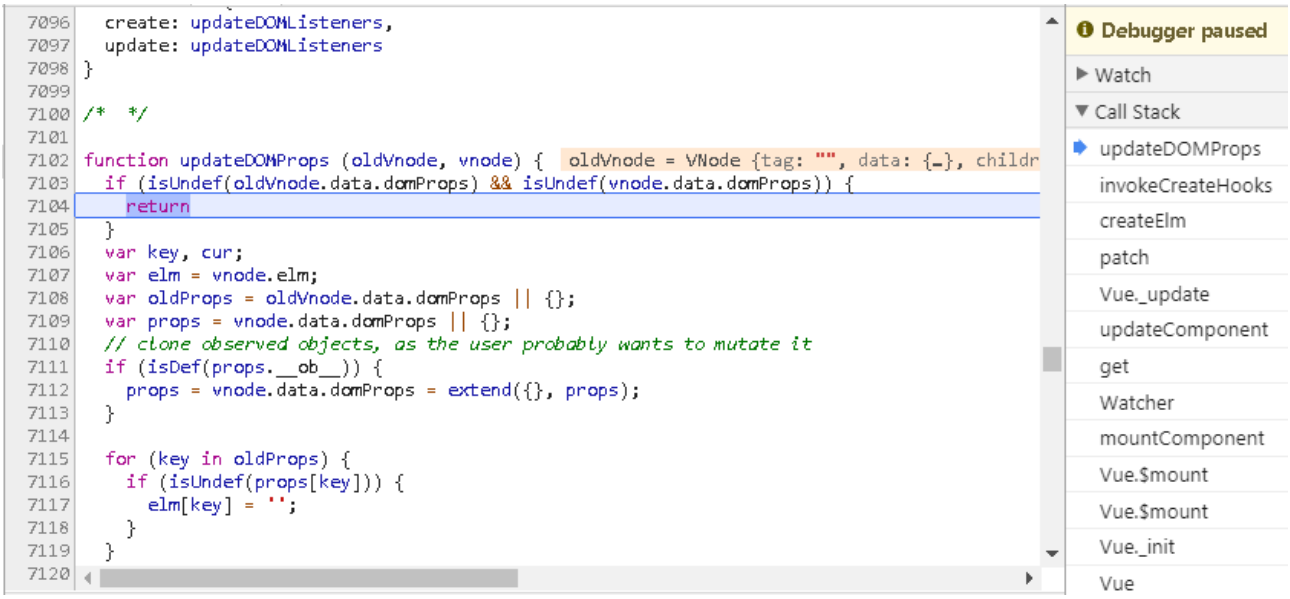


134. Retornamos a ***invokeCreateHooks*** :

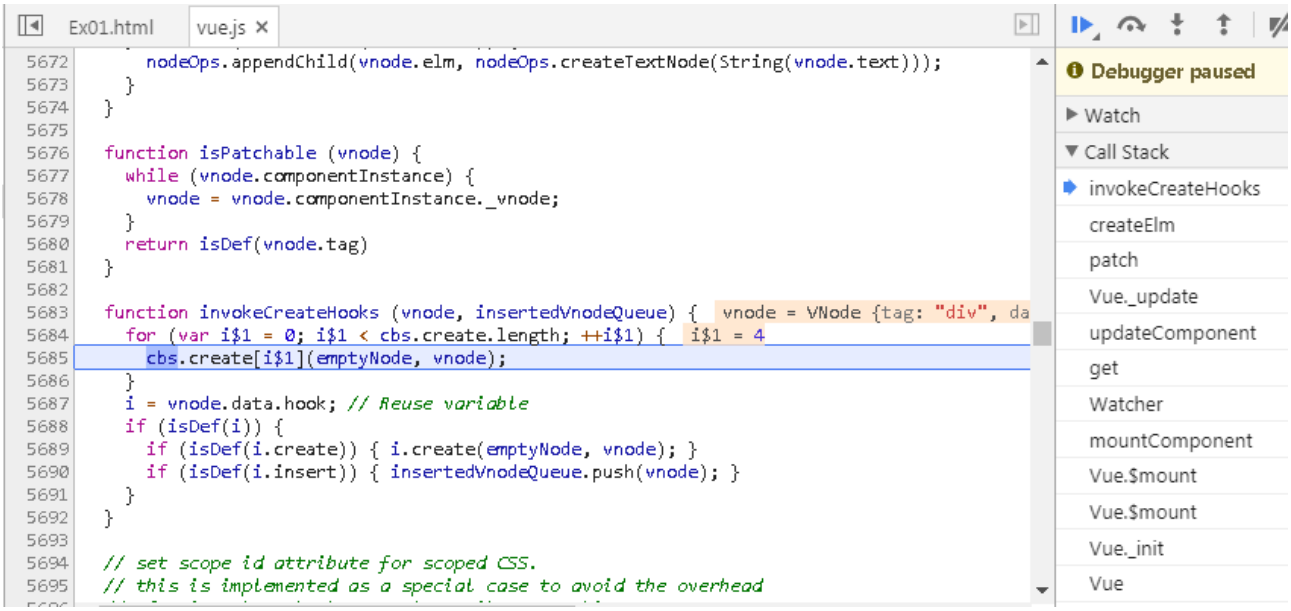


VUE – Substituição de variáveis

135. Agora é chamado o hook de ***updateDOMProps*** :

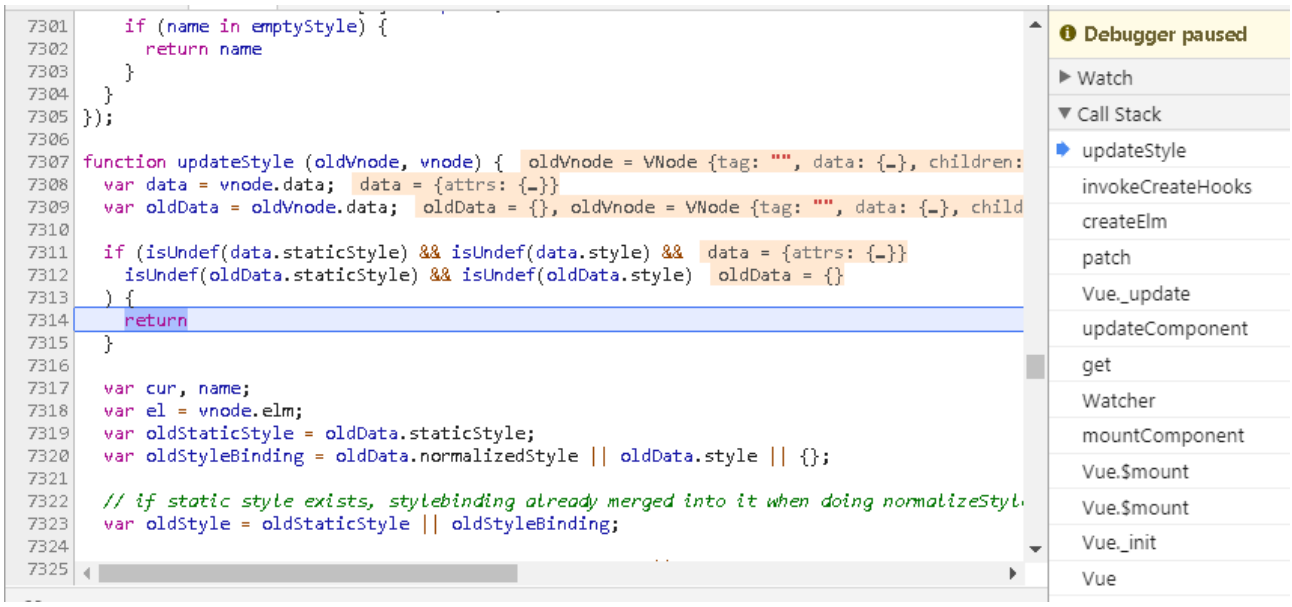


136. Retornamos a ***invokeCreateHooks*** :

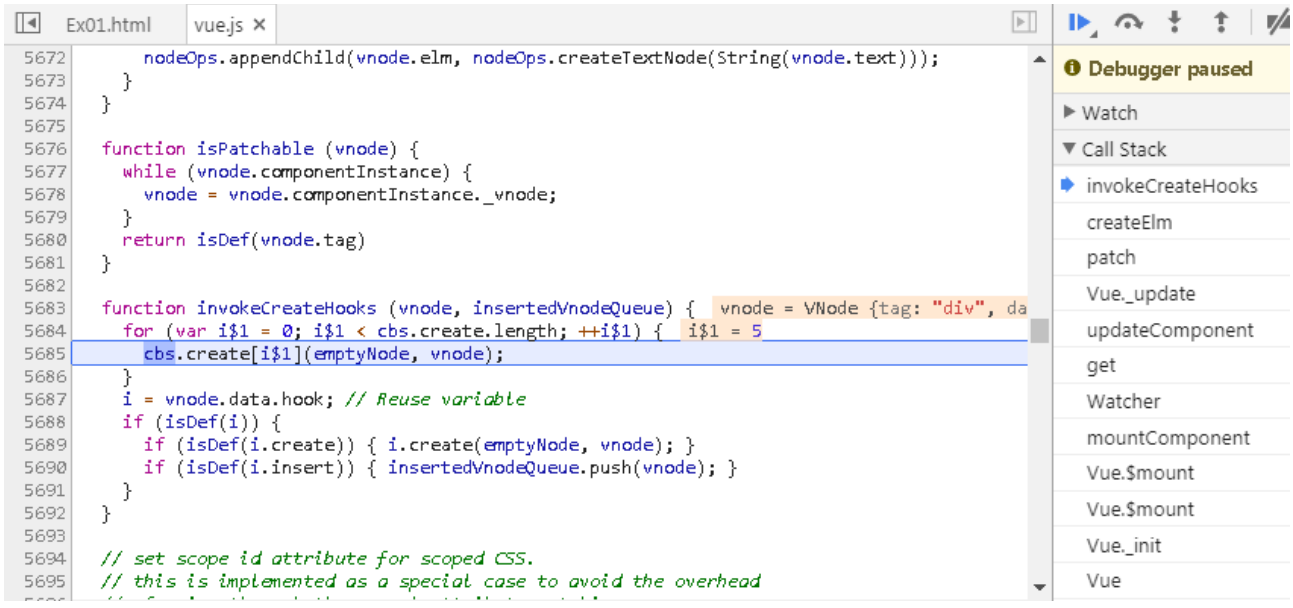


VUE – Substituição de variáveis

137. Agora é chamado o hook de ***updateStyle*** :

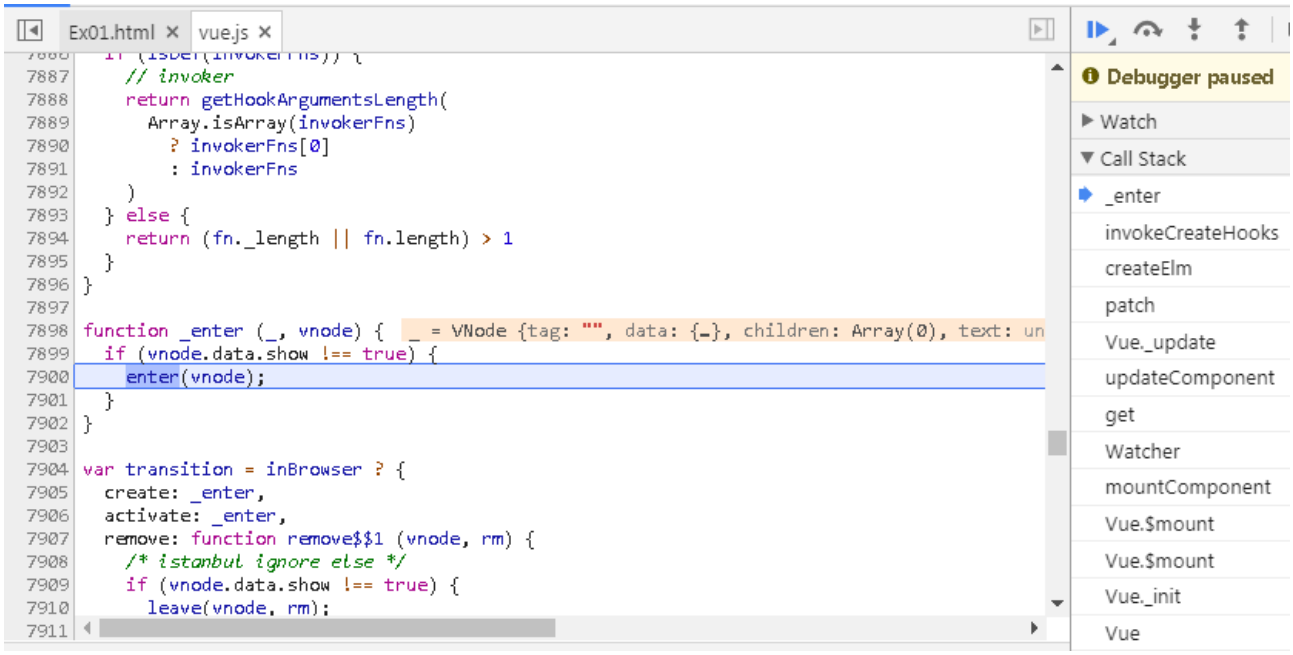


138. Retornamos a ***invokeCreateHooks*** :

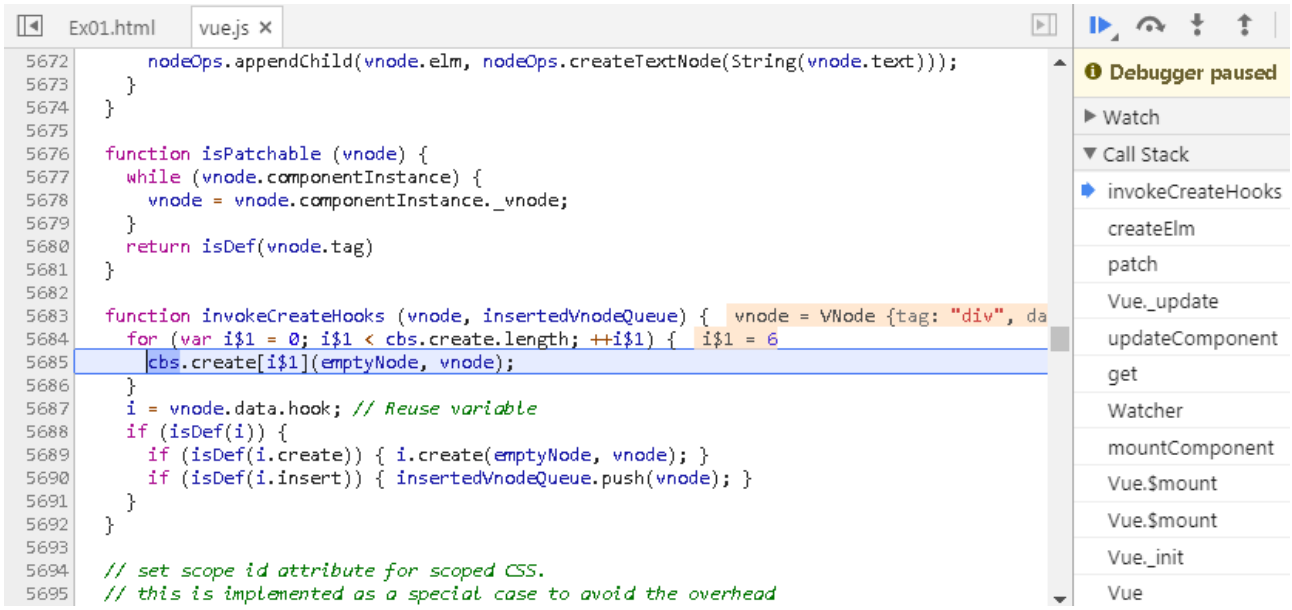


VUE – Substituição de variáveis

139. Agora é chamado o hook de ***enter*** :

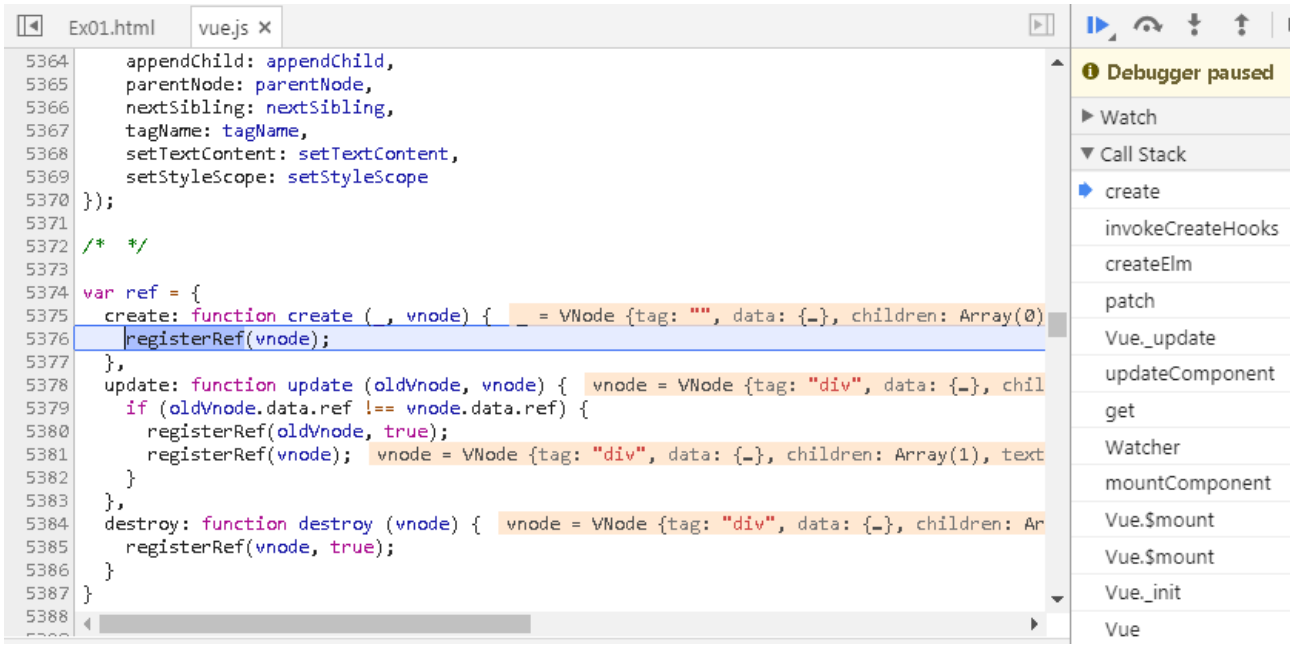


140. Retornamos a ***invokeCreateHooks*** :

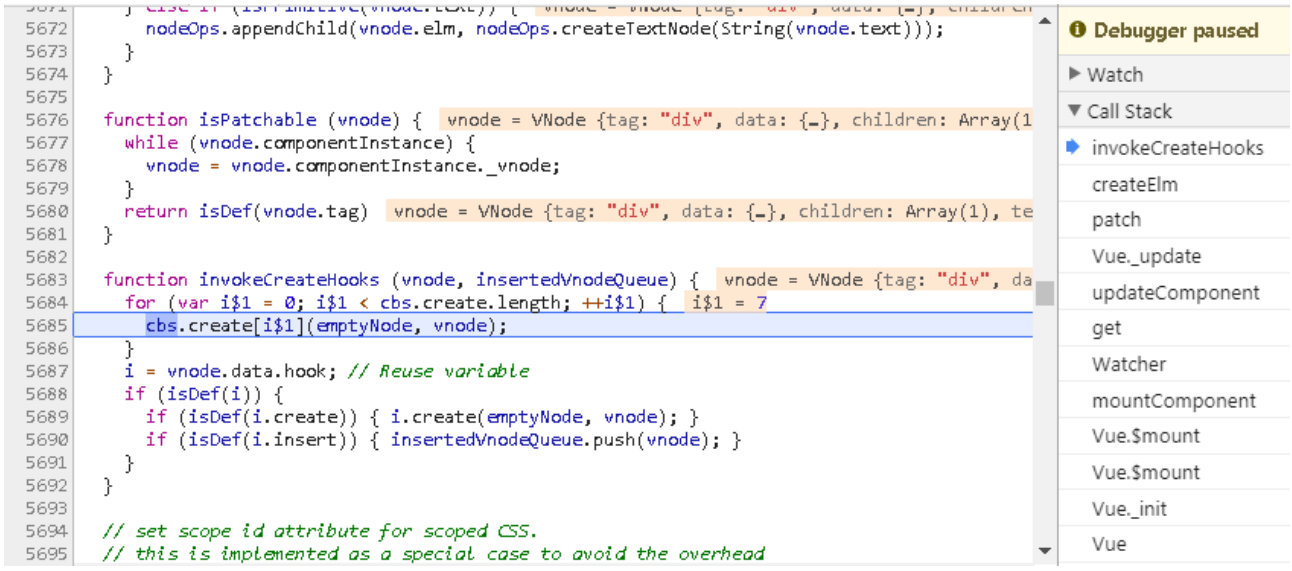


VUE – Substituição de variáveis

141. A seguir a execução do hook ***create***:

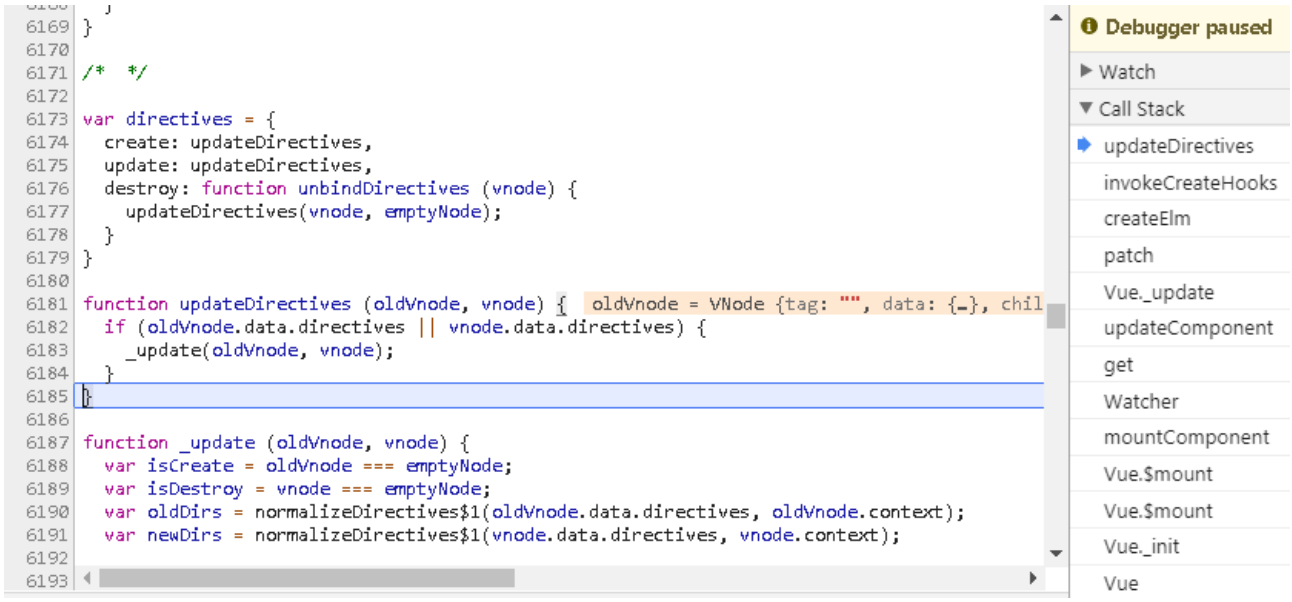


142. Retornamos a ***invokeCreateHooks*** :

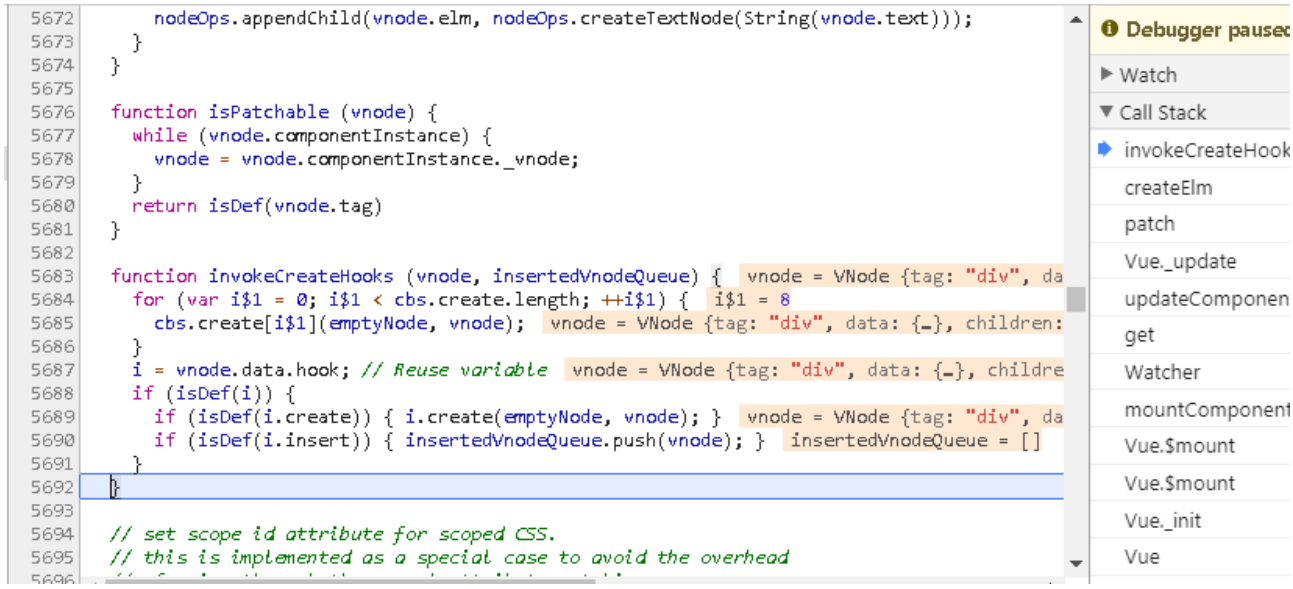


VUE – Substituição de variáveis

143. Agora a chamada à ***updateDirectives***:

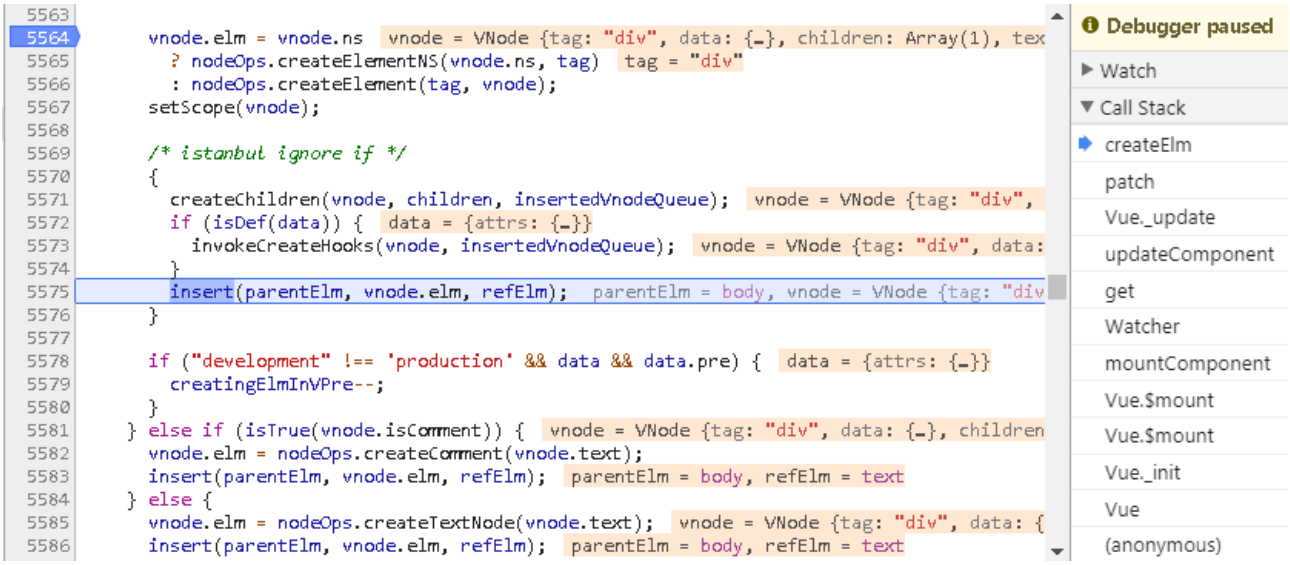


144. Finalizamos ***invokeCreateHooks*** :

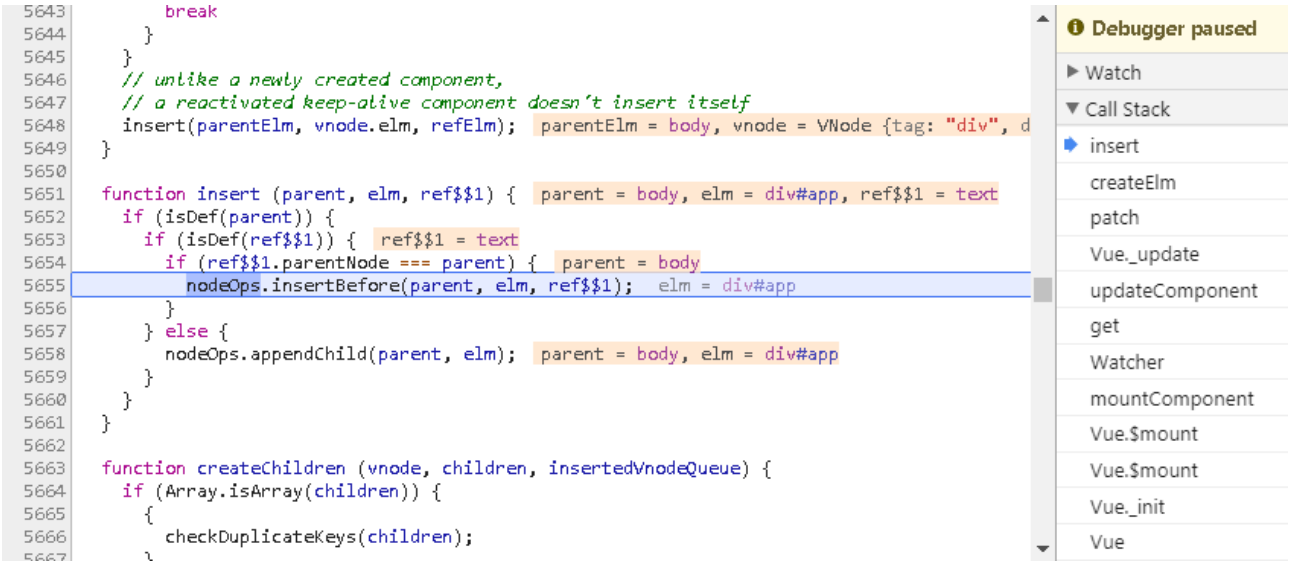


VUE – Substituição de variáveis

145. Retorno à ***createElm.*** Depois de ***invokeCreateHooks***, ***createElm*** chama ***insert***:



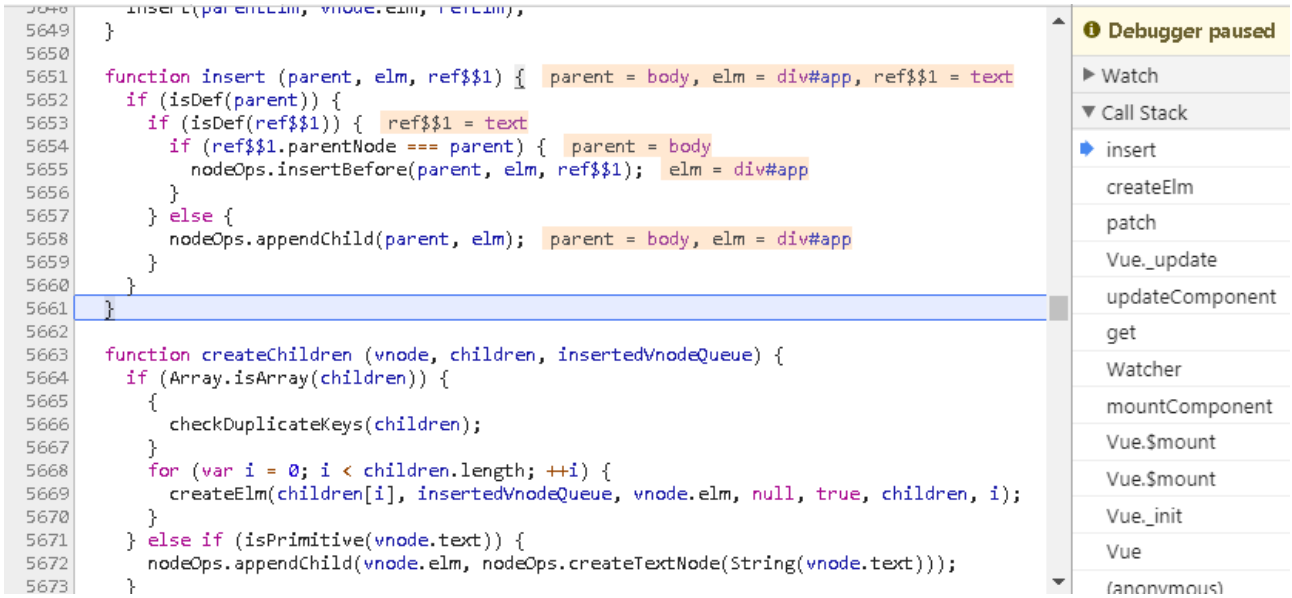
146. A função ***insert*** chama o método ***insertBefore*** de **nodeOps**:



VUE – Substituição de variáveis

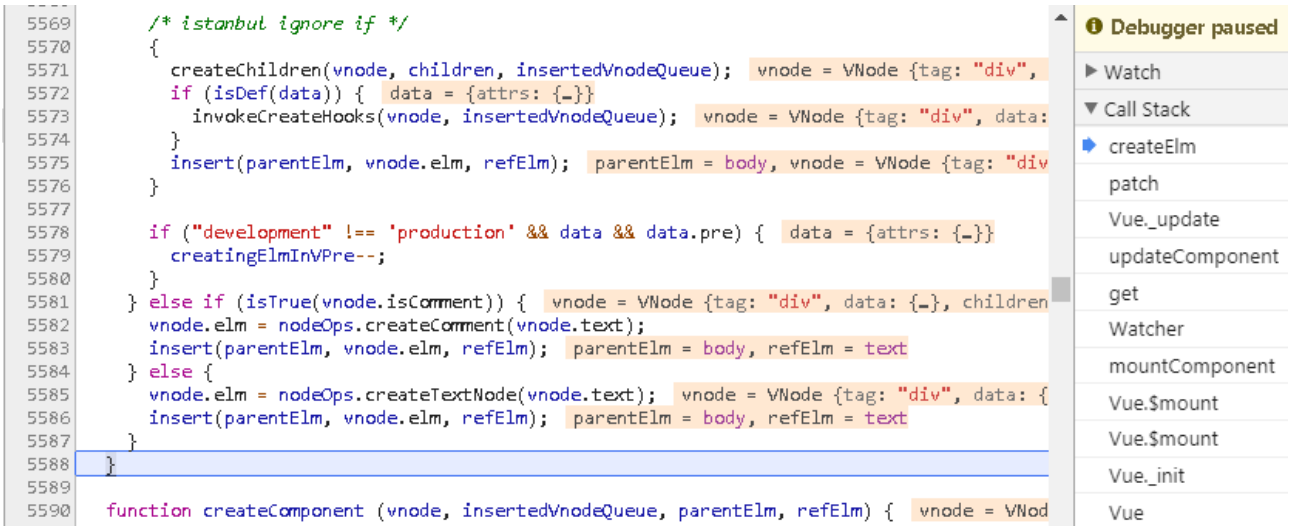
147. A função ***insertBefore*** chama o método ***insertBefore*** do objeto Node:

148. A função chamada se encerra:

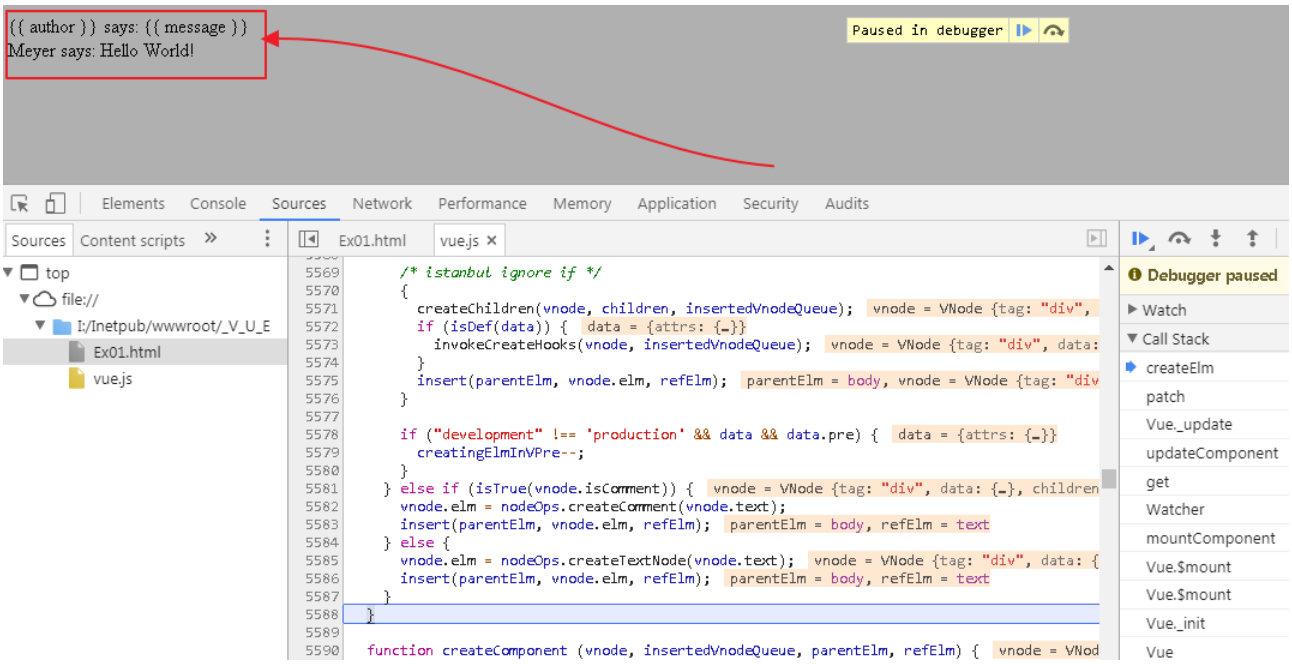


VUE – Substituição de variáveis

149. A função ***createElm*** chega ao fim:

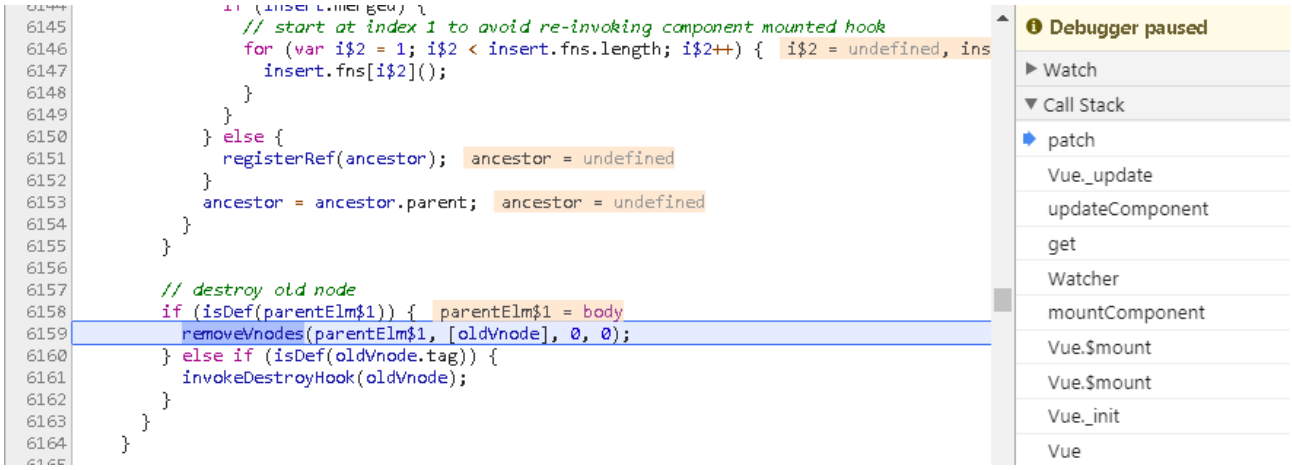


150. O nodo inserido convive, por alguns segundos, com o nó que sustenta a template. A primeira linha representa a template em termo de variáveis VUE. A segunda linha mostra justamente um nó resultante de todo este processamento da função de renderização, produzida pelo processo de compilação.

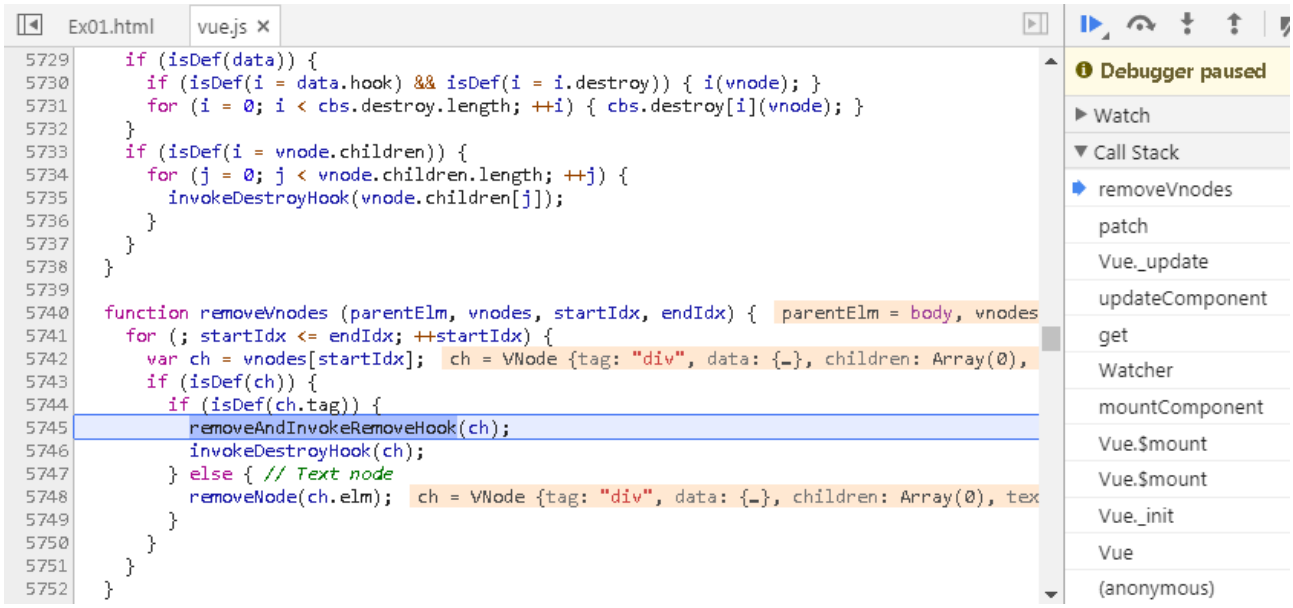


VUE – Substituição de variáveis

151. Será preciso remover o nodo que contém a template. Isto é feito pela função ***removeVnodes***:

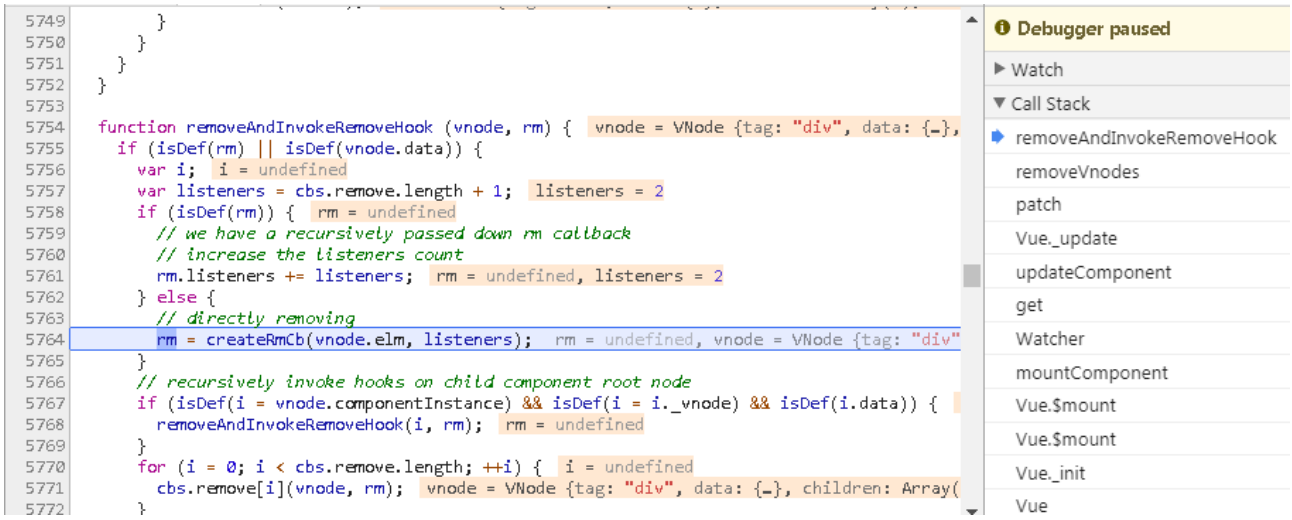


152. A função ***removeVnodes*** invoca ***removeAndInvokeRemoveHook***:

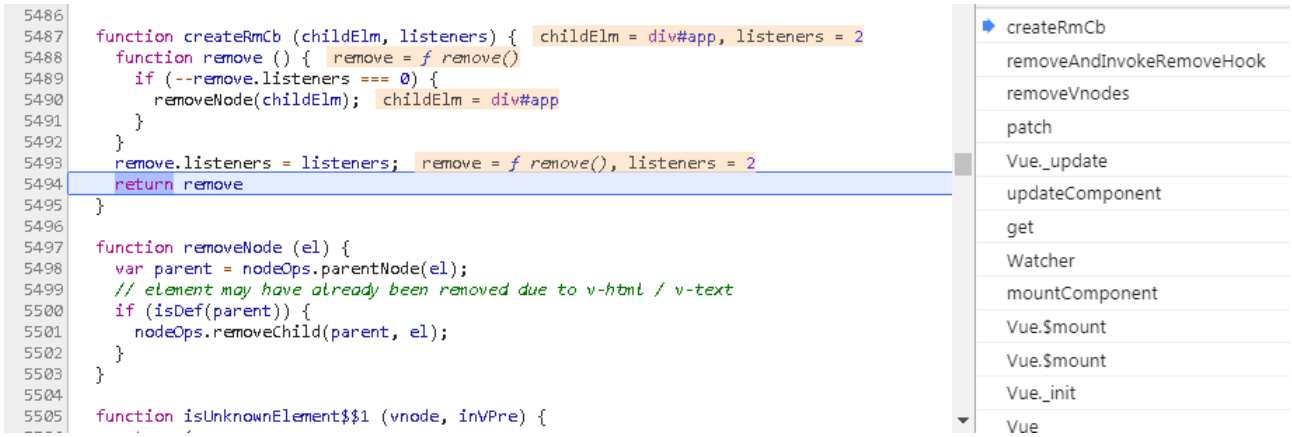


VUE – Substituição de variáveis

153. Que, por sua vez, chama ***createRmCb***:

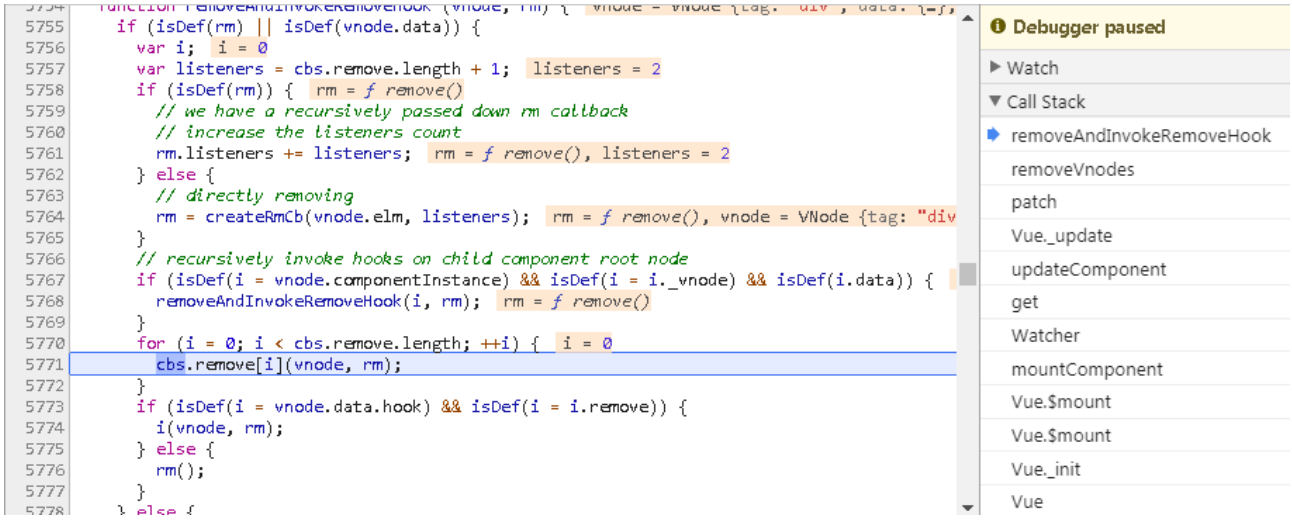


154. A função retorna **remove**:



VUE – Substituição de variáveis

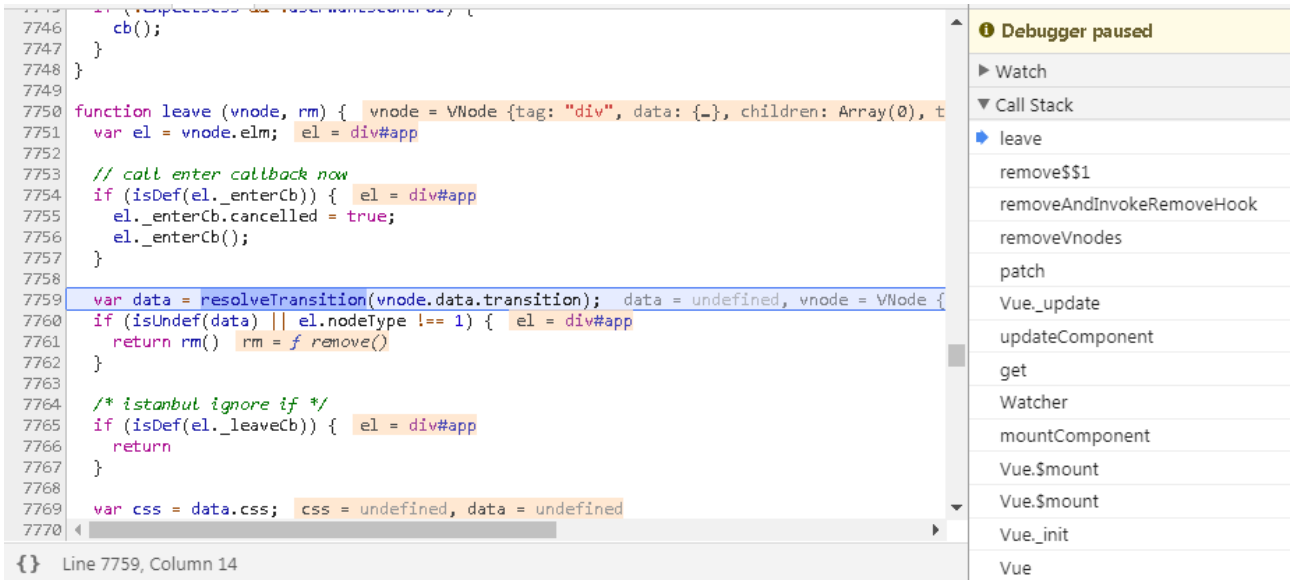
155. Retornamos a removeAndInvokeRemoveHook:



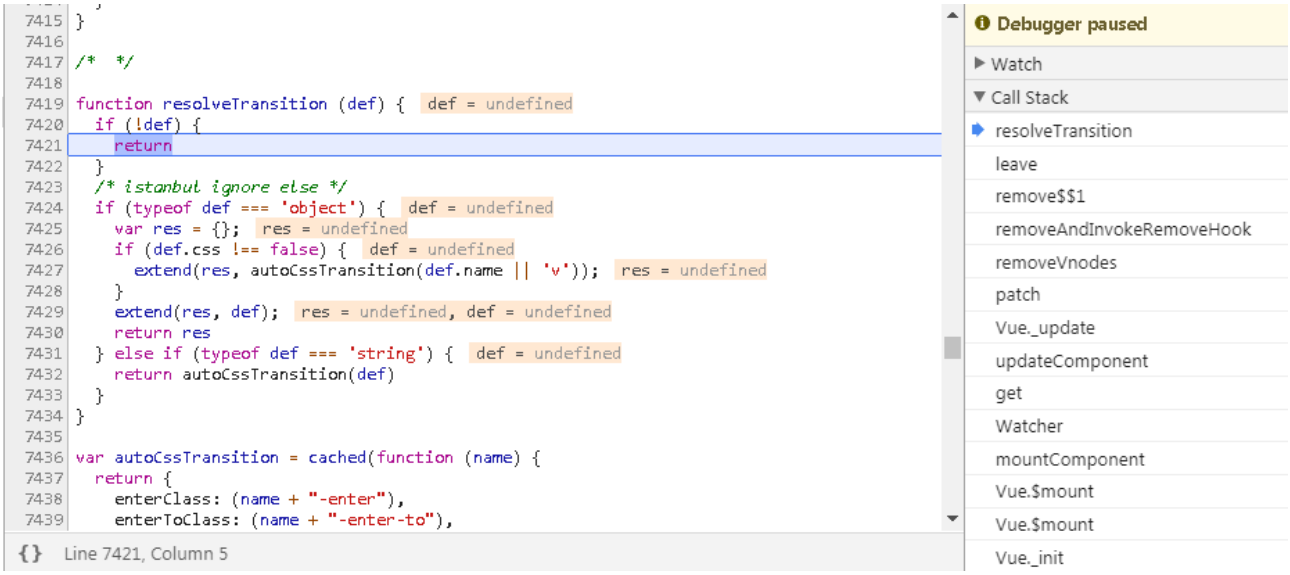
156. A chamada aponta para a função definida no array de hooks do VUE (***remove$$1***). Esta chama a função ***leave***:

VUE – Substituição de variáveis

157. A função leave, entre outras coisas, chama ***resolveTransition***:

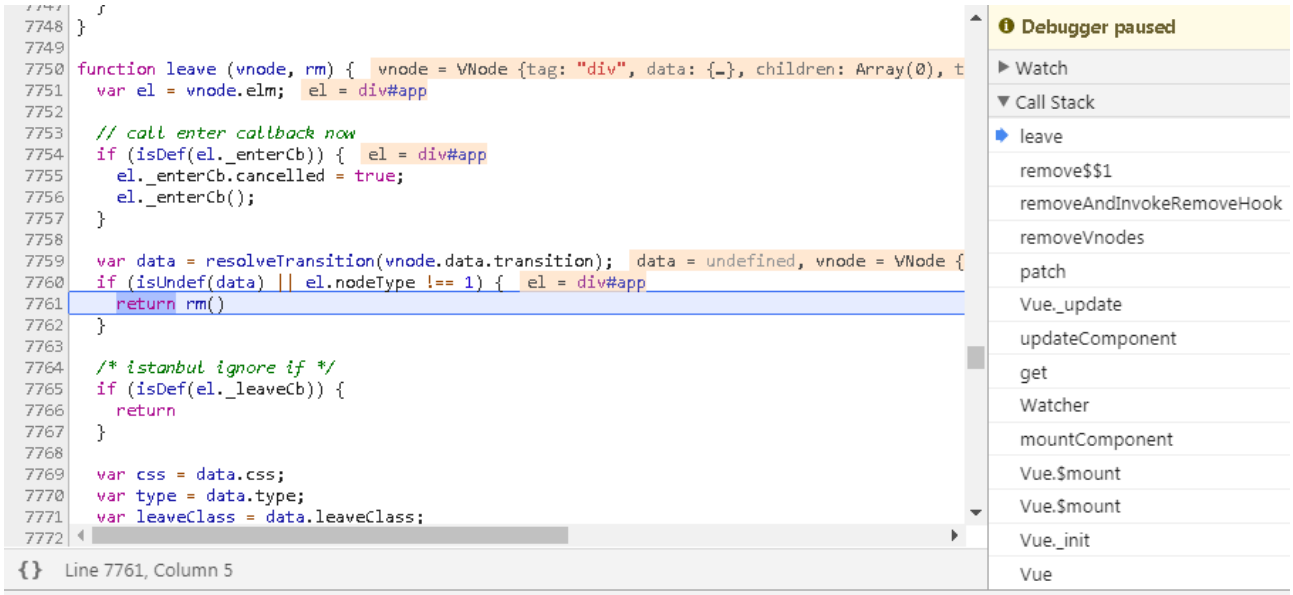


158. que apenas faz um teste do objeto **def**:



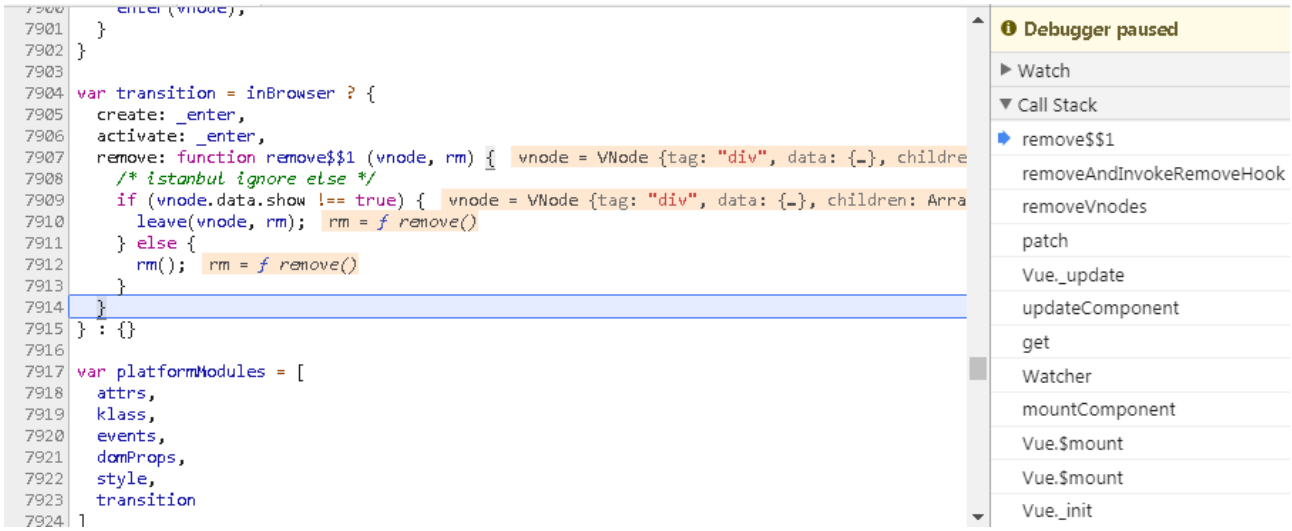
VUE – Substituição de variáveis

159. Retornamos à função ***leave***:



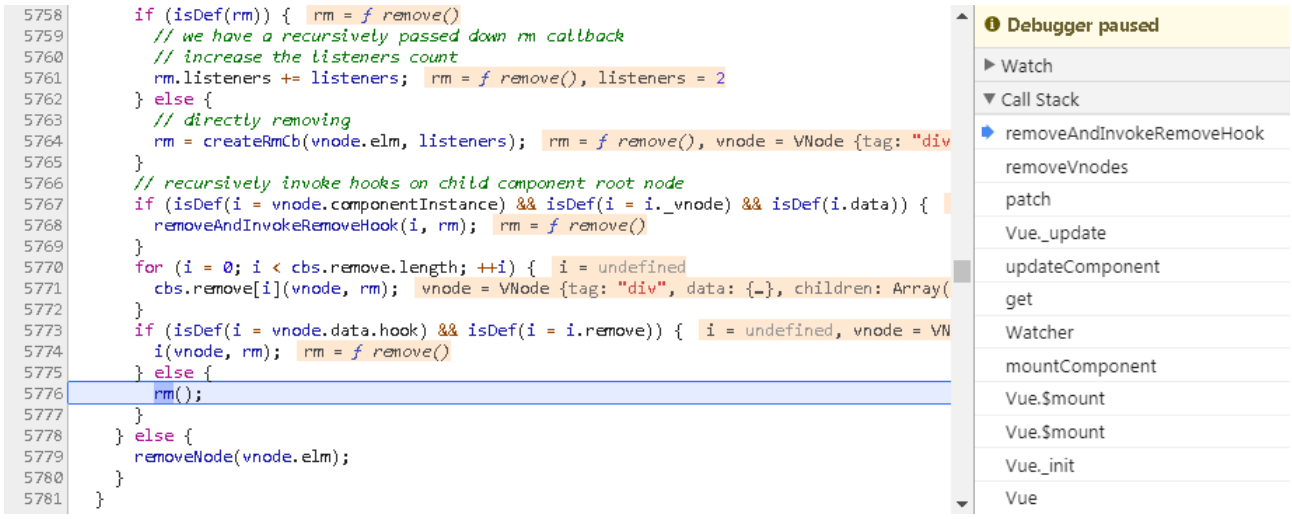
que retorna o valor da função ***rm***.

160. A função aponta para remove$$1:

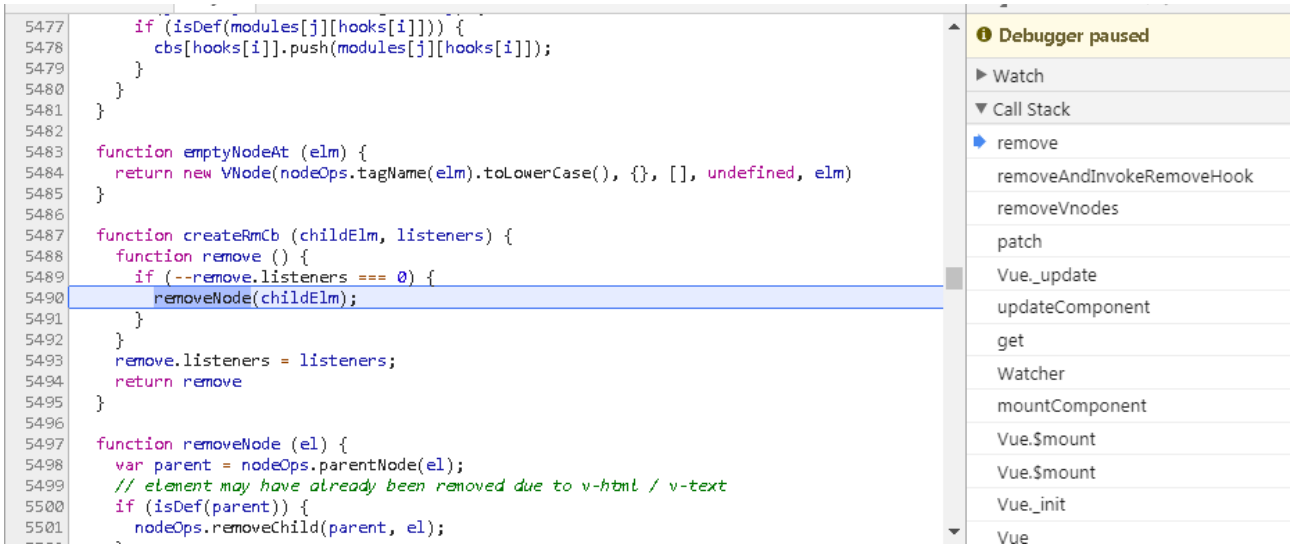


VUE – Substituição de variáveis

161. Retornamos a removeAndInvokeRemoveHook:

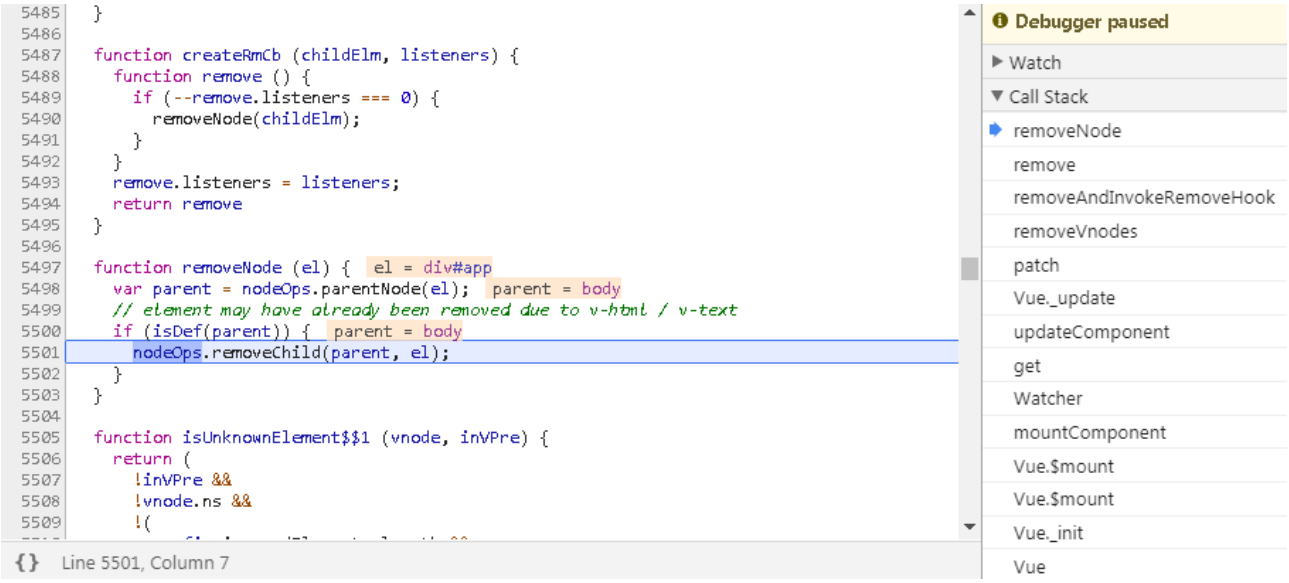


162. E ***rm*** desvia o processamento para a função ***remove***, definida no corpo de ***createRmCb***:

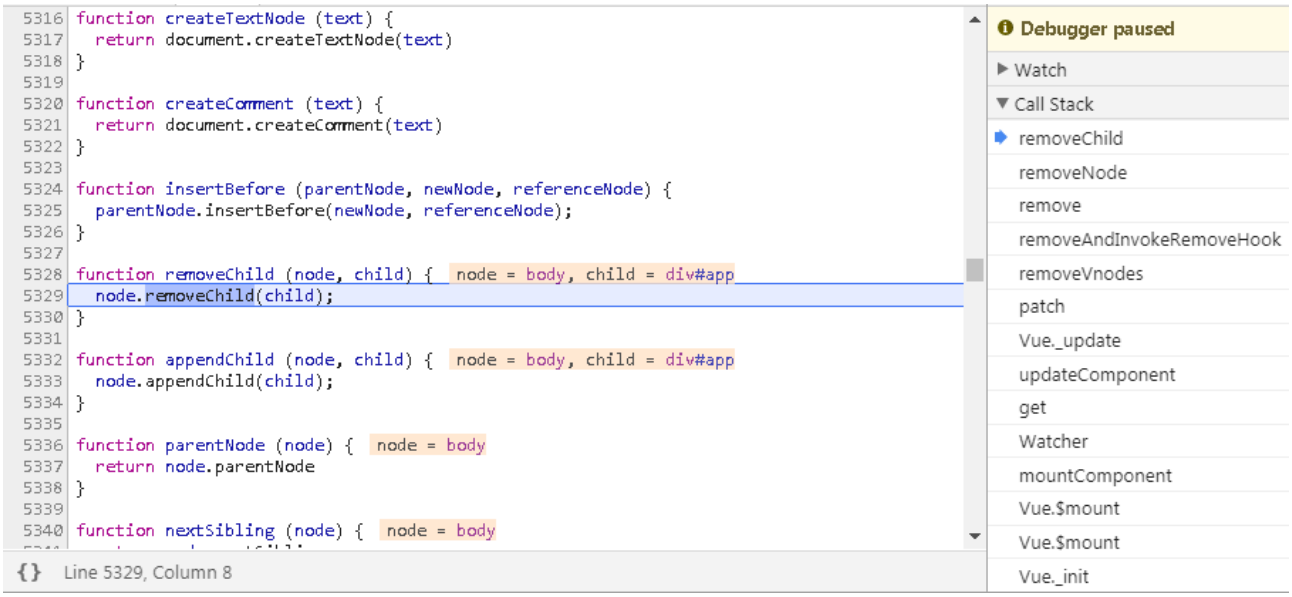


VUE – Substituição de variáveis

163. Retornamos a ***removeNode***, que chama o método removeChild do objeto nodeOps:

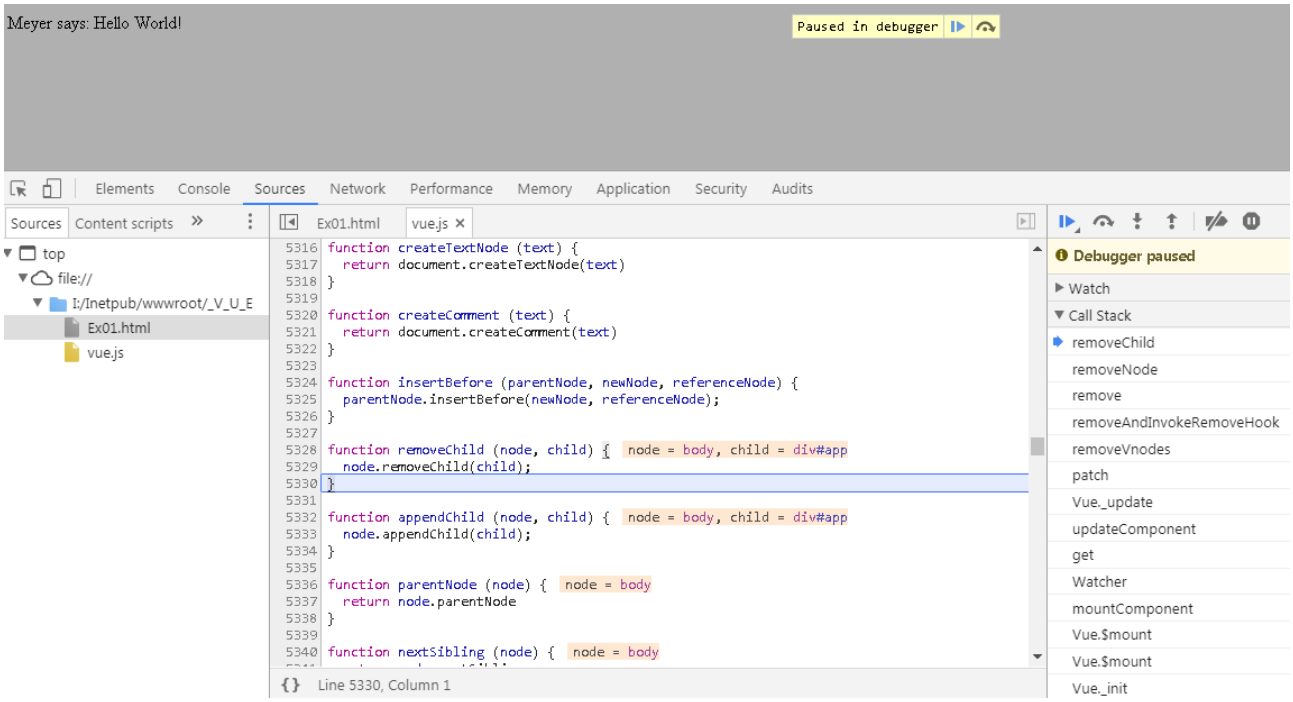


164. Este método chama o método de mesmo nome do objeto **node** do DOM:

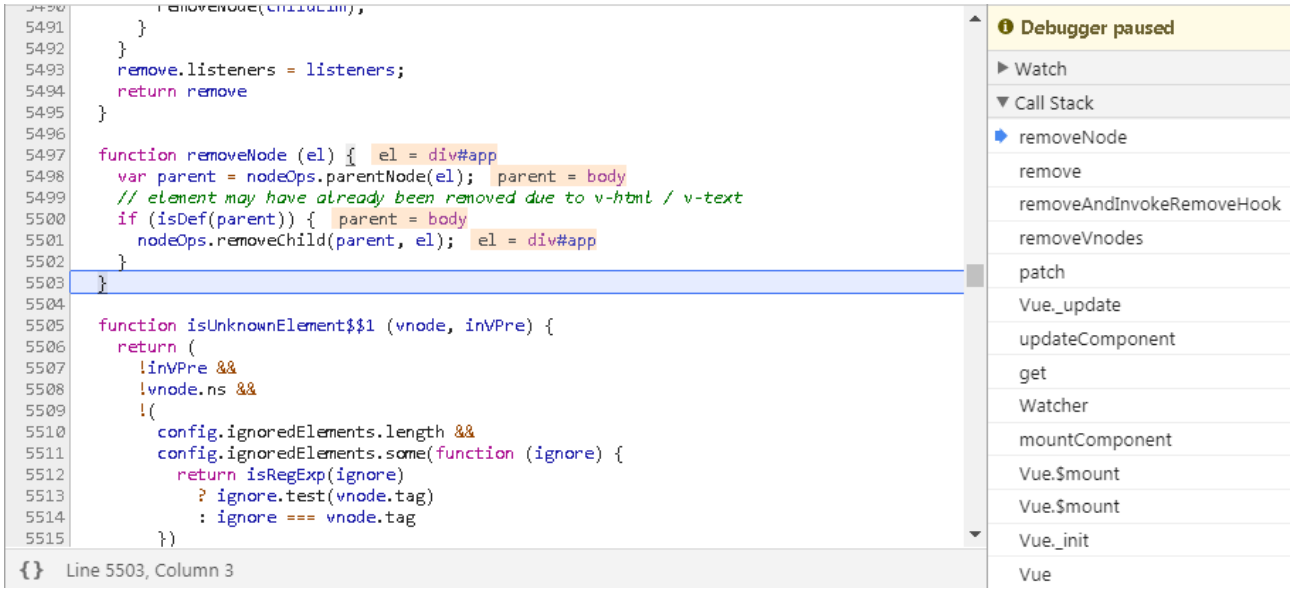


VUE – Substituição de variáveis

165. A execução deste método é que faz sumir o nodo da template, que conviveu, por alguns segundos com o resultado da função de renderização:

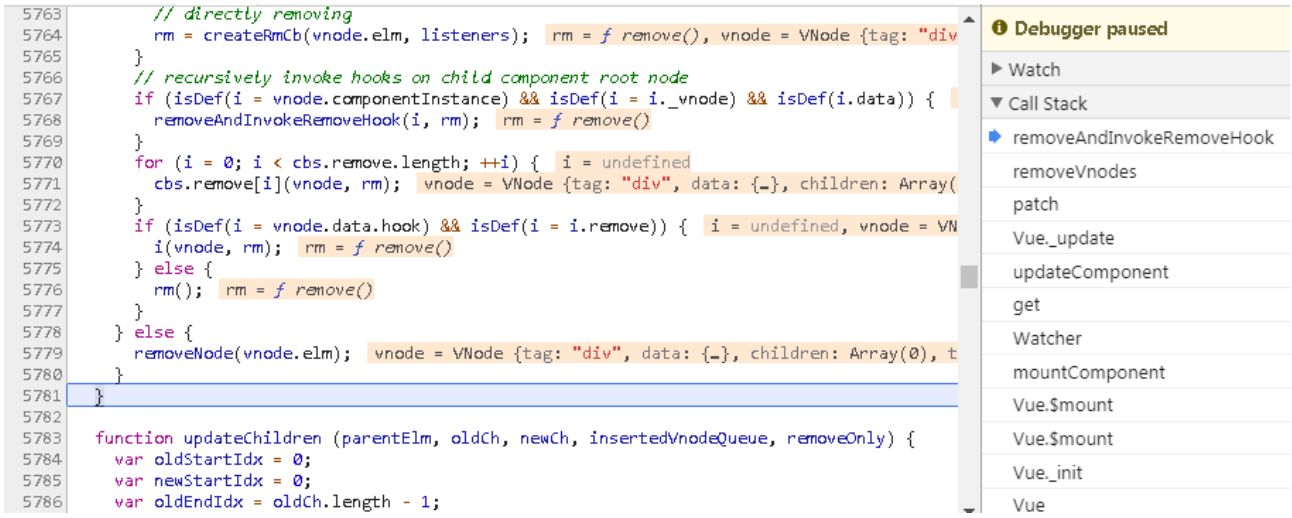


166. Retornamos a ***removeNode***:

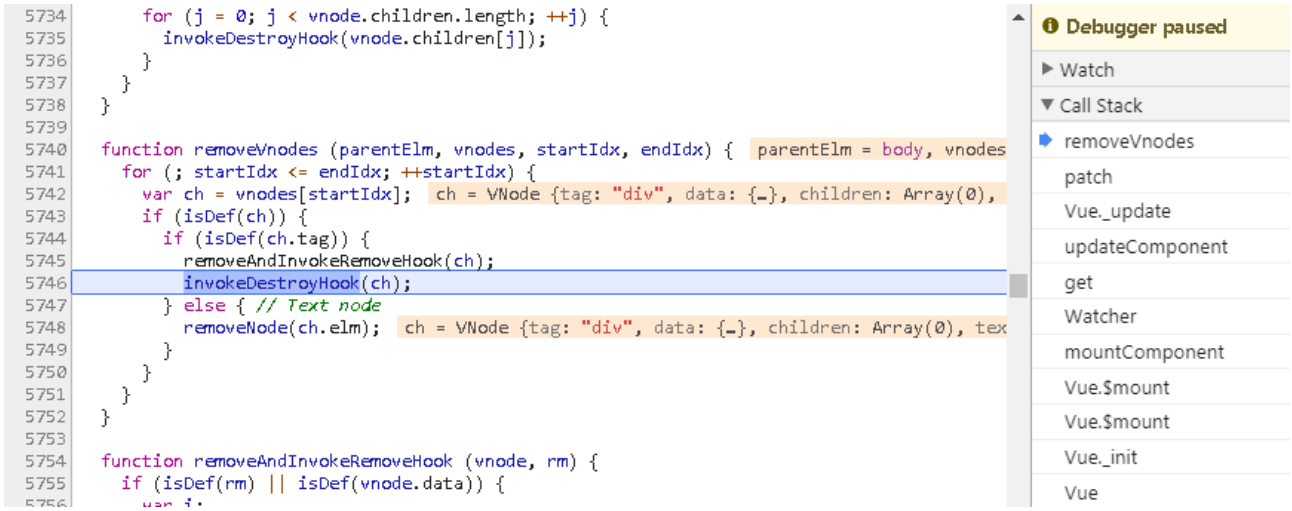


VUE – Substituição de variáveis

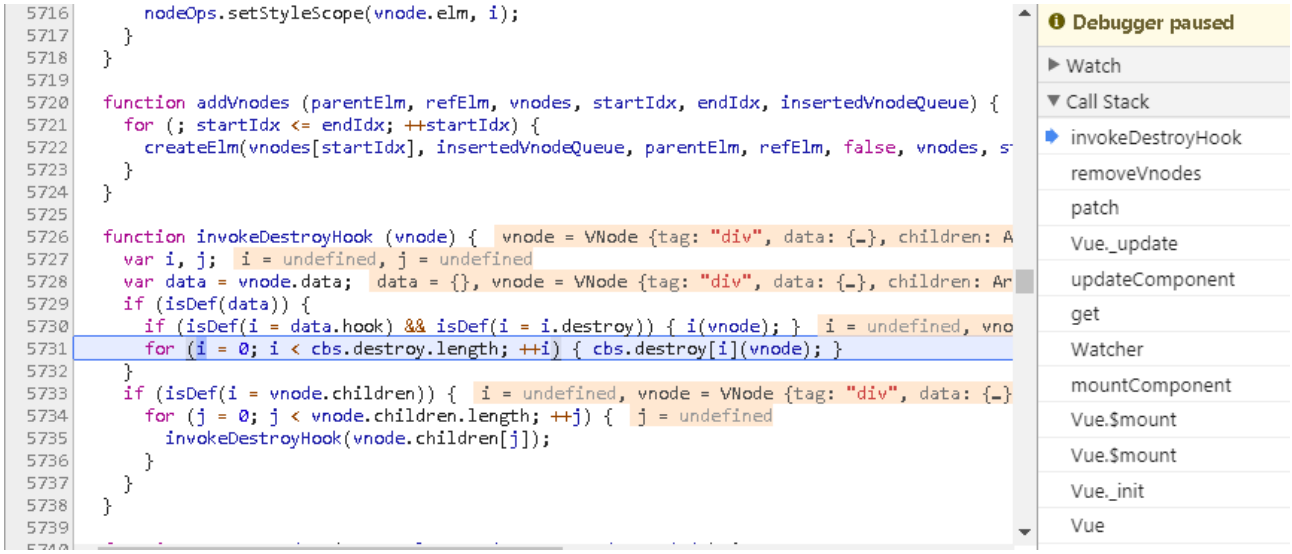
167. Retornamos a ***removeAndInvokeRemoveHook***:



168. Em ***removeVnodes*** é chamada ***invokeDestroyHook***:



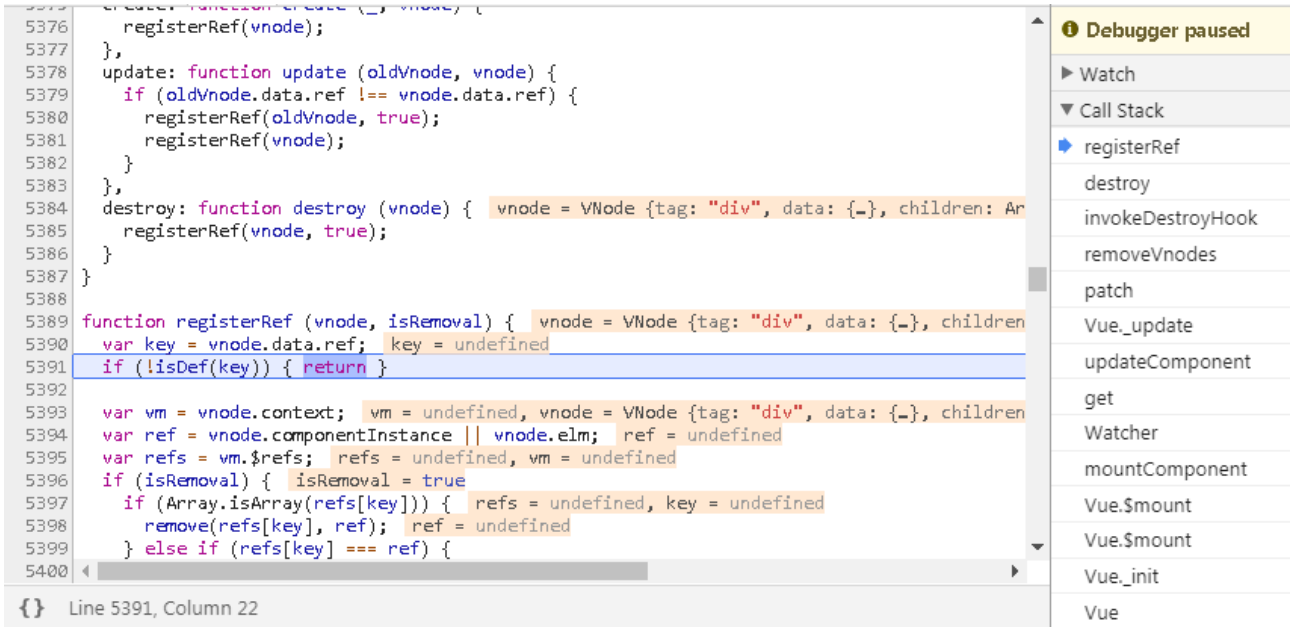
169. Esta função chama os hooks do tipo destroy enumerados no objeto **cbs**:



VUE – Substituição de variáveis

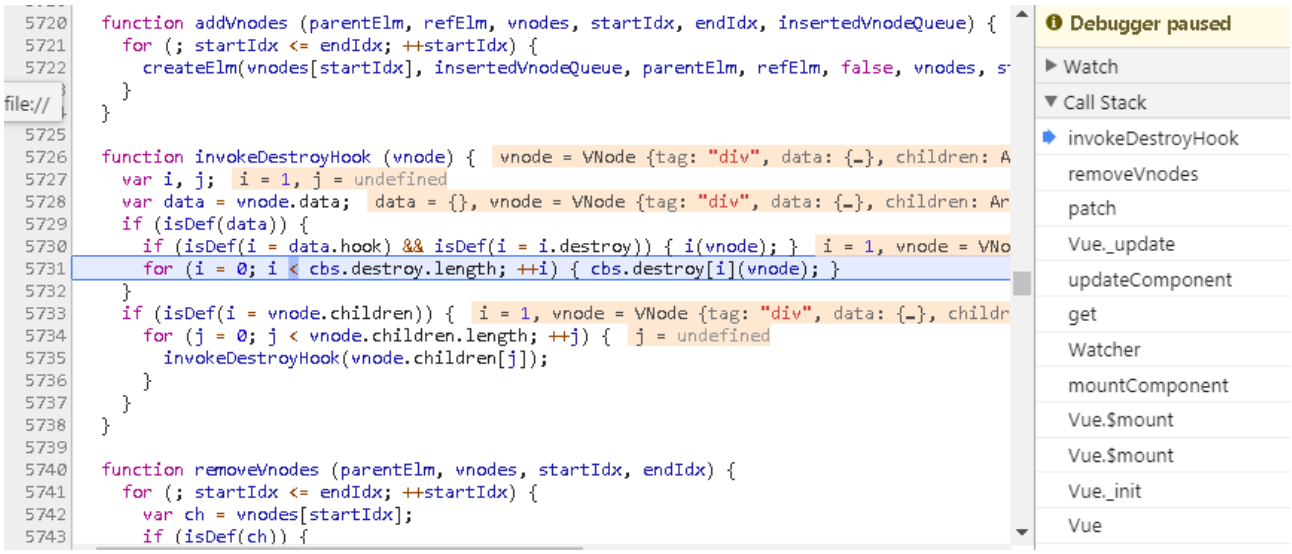
170. Um hook aponta para o método ***destroy*** do objeto **ref**:. Este chama ***registerRef***:

171. A função ***registerRef*** toma suas providências:

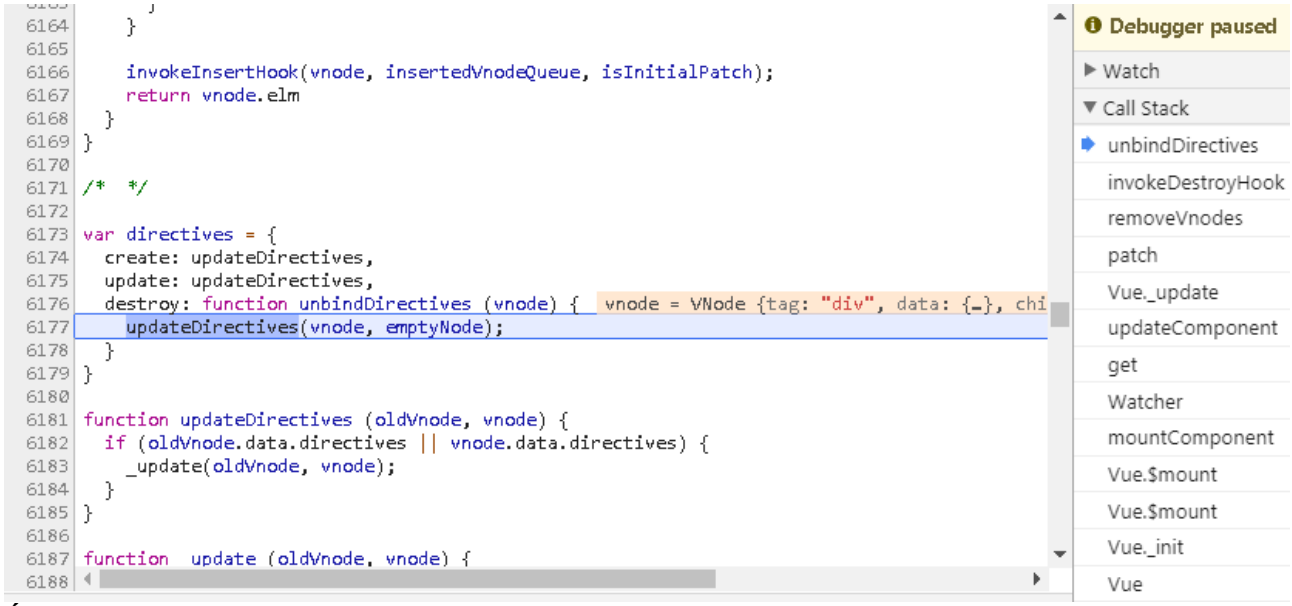


VUE – Substituição de variáveis

172. Retornamos à ***invokeDestroyHook***:



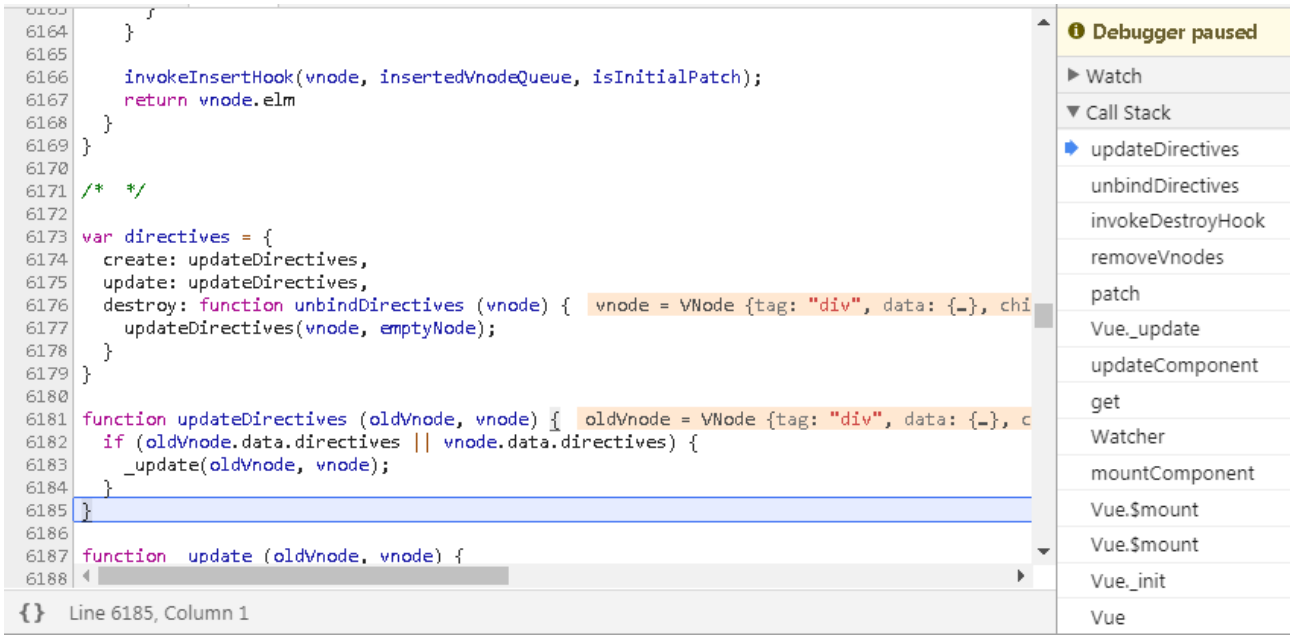
173. Entre os hooks do tipo destroy está ***unbindDirectives***:



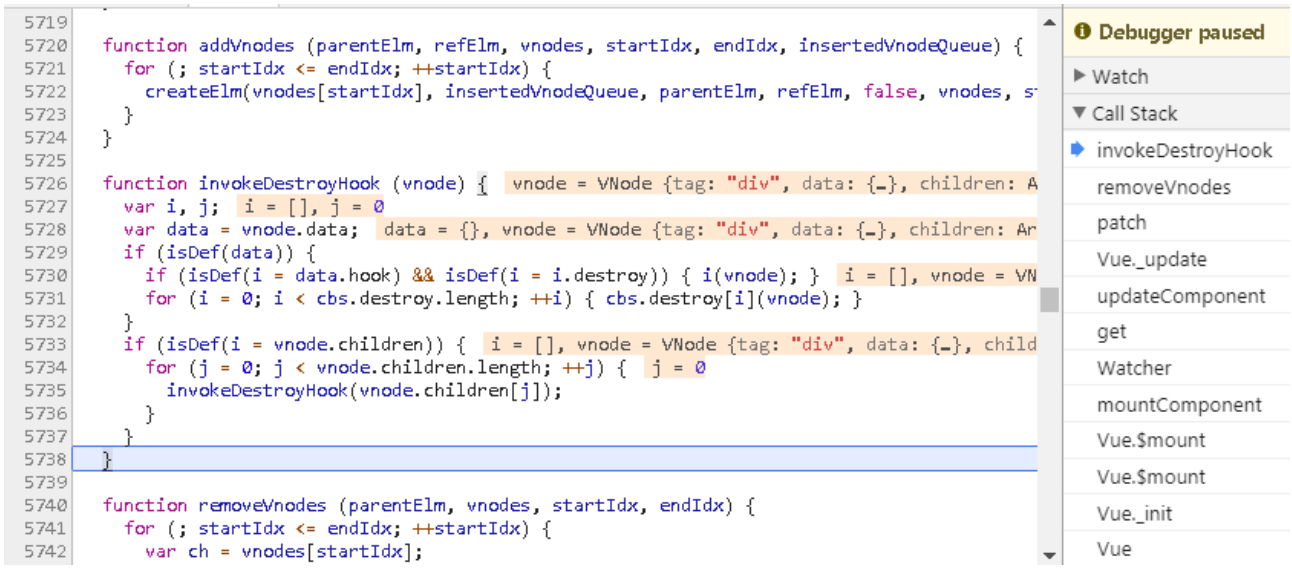
É chamada a função ***updateDirectives***.

VUE – Substituição de variáveis

174. Fim de ***updateDirectives***:

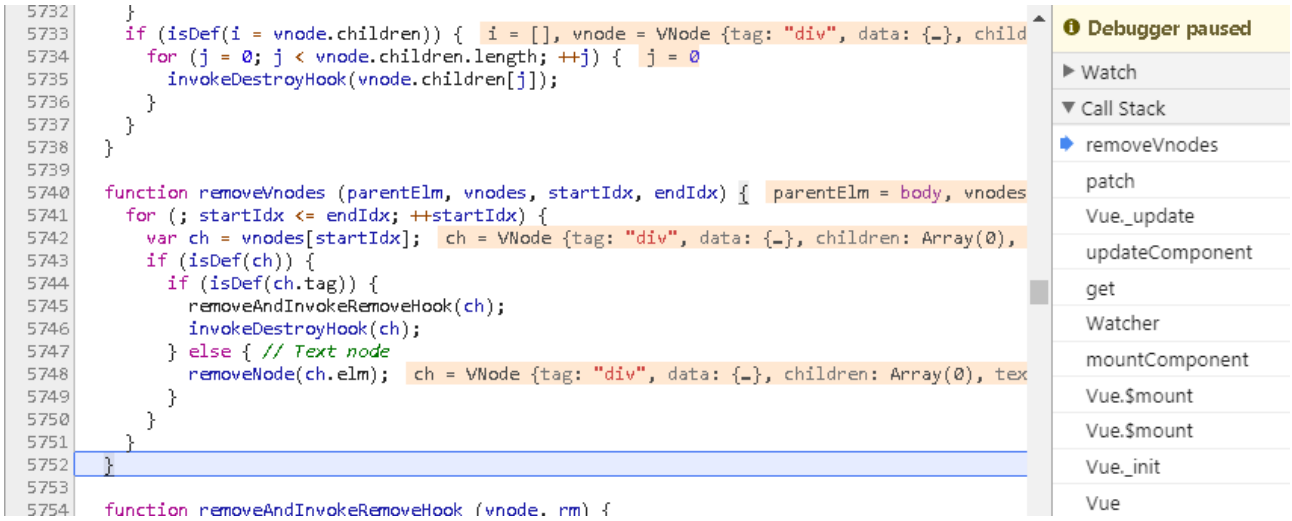


175. Retorno a invokeDestroyHook. Preparando para retorno:

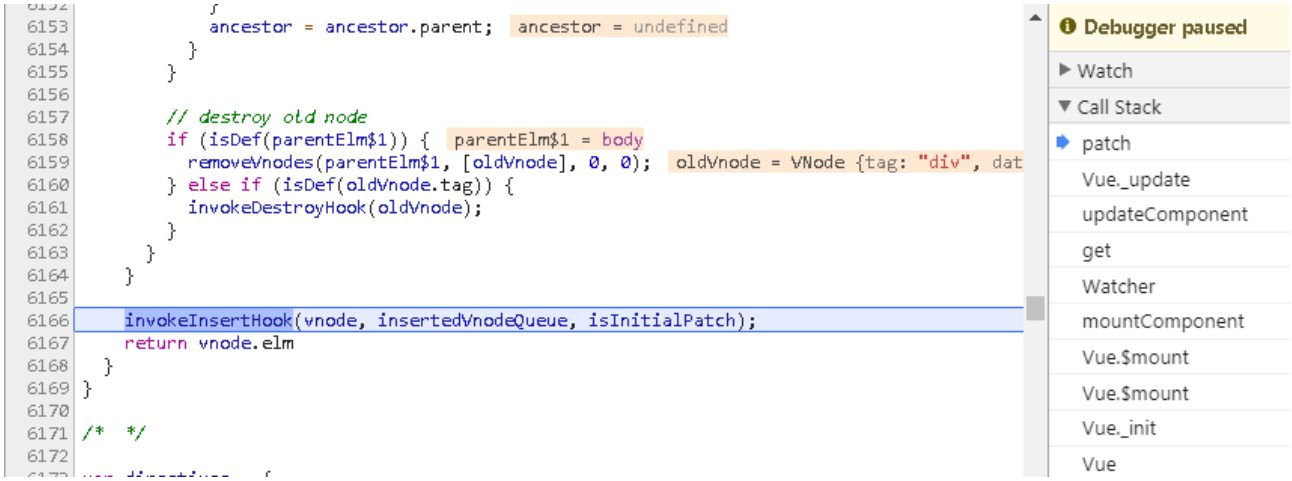


VUE – Substituição de variáveis

176. Retorno a removeVnode e preparação para sair:

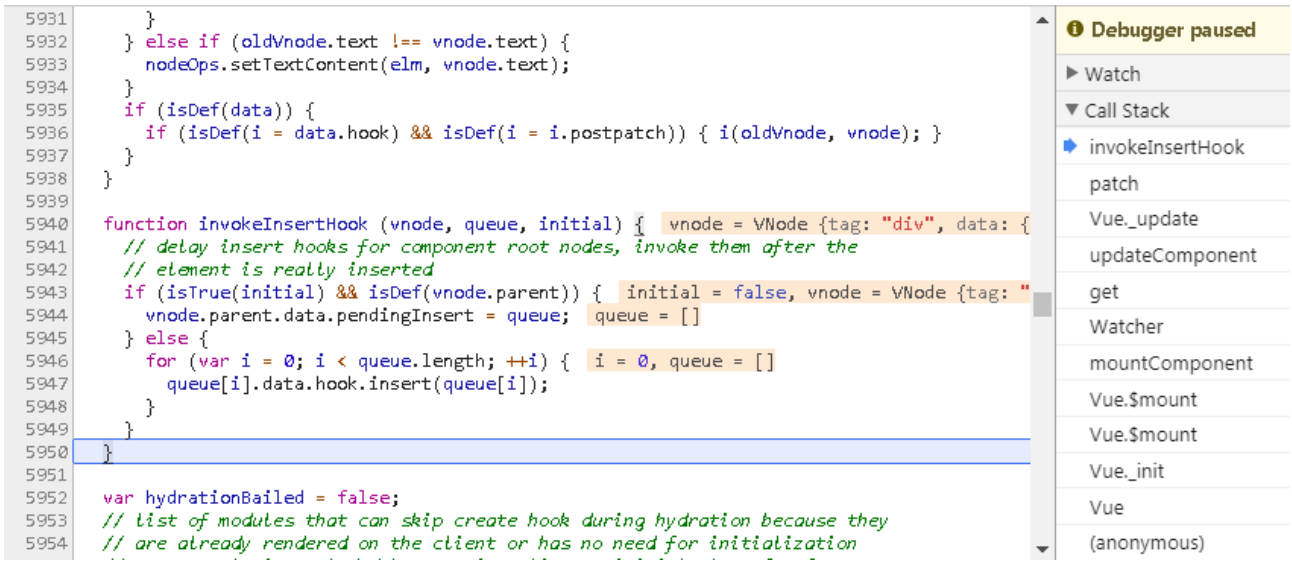


177. Retorno à ***patch***:

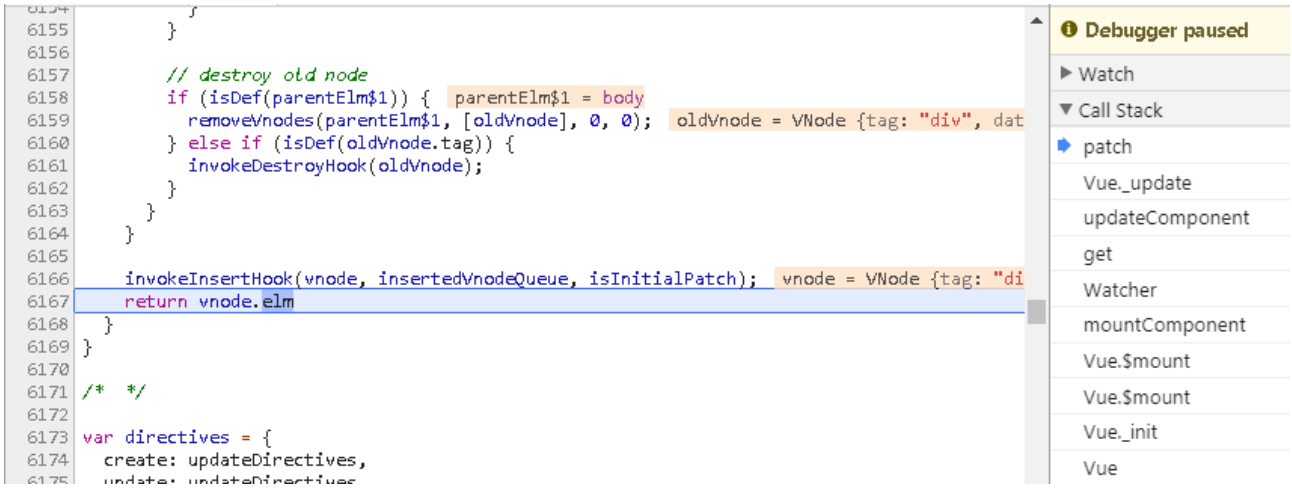


VUE – Substituição de variáveis

178. Retorno à ***invokeInsertHook*** e preparação para retorno:

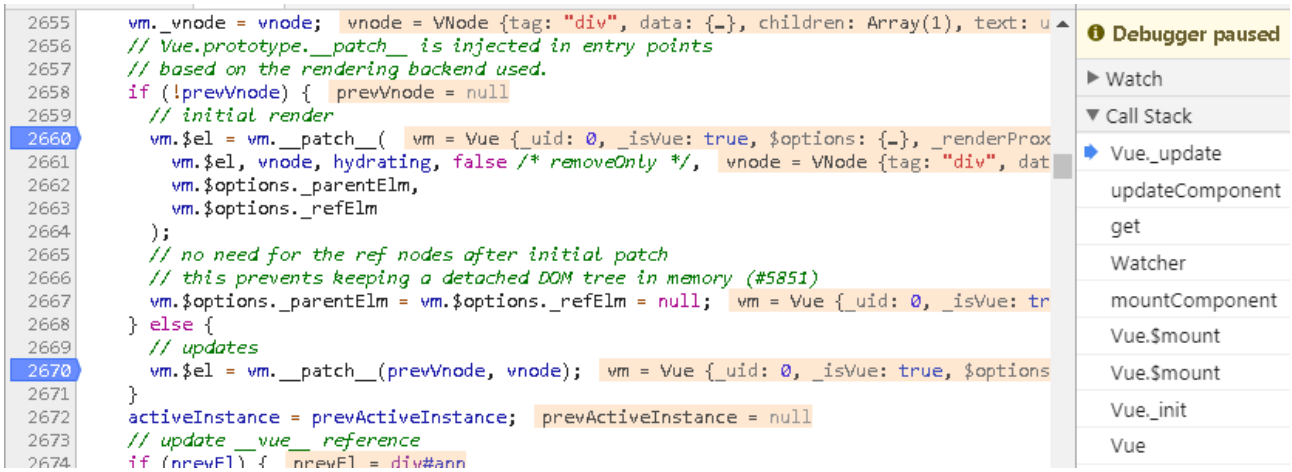


179. retorno à função ***patch***:

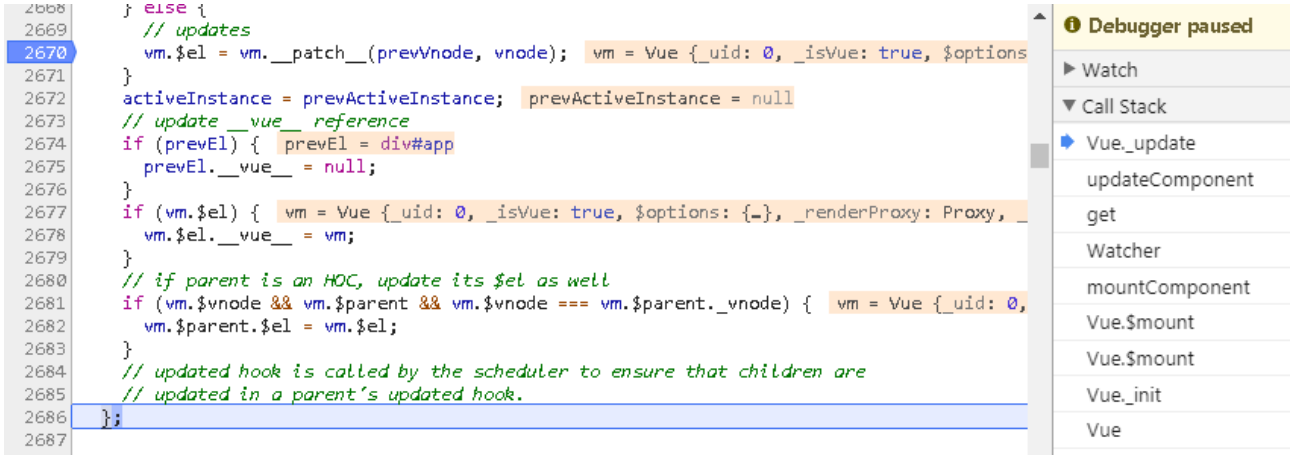


VUE – Substituição de variáveis

180. Retorno à ***Vue._update***:

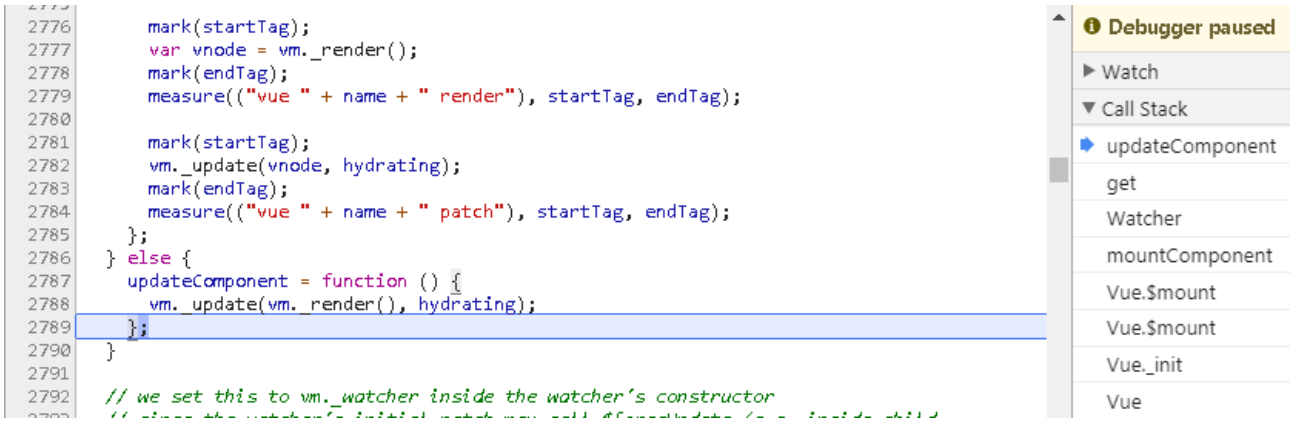


181. Preparando para retorno:

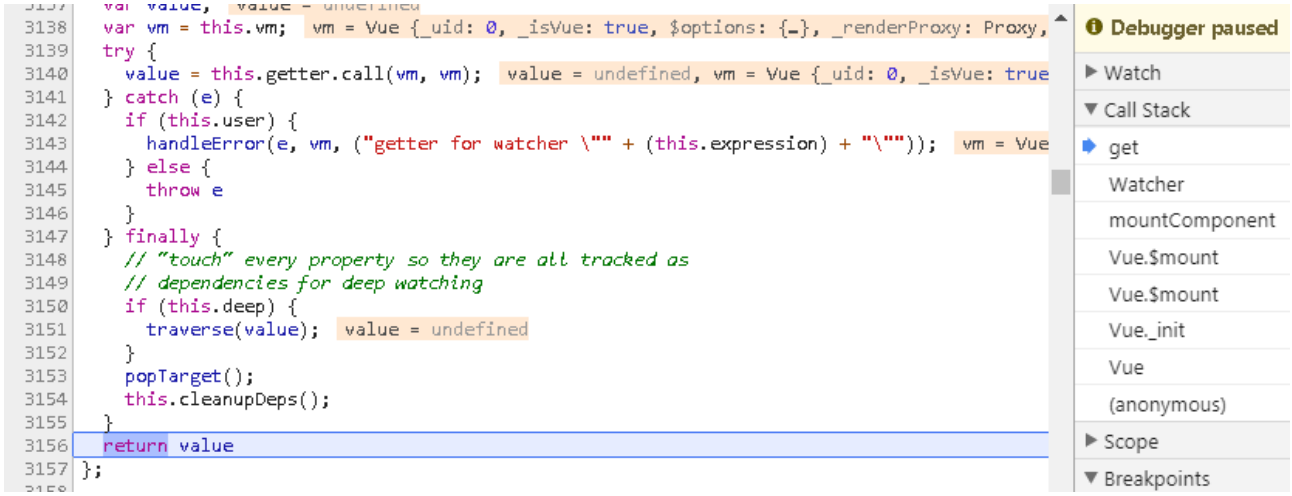


VUE – Substituição de variáveis

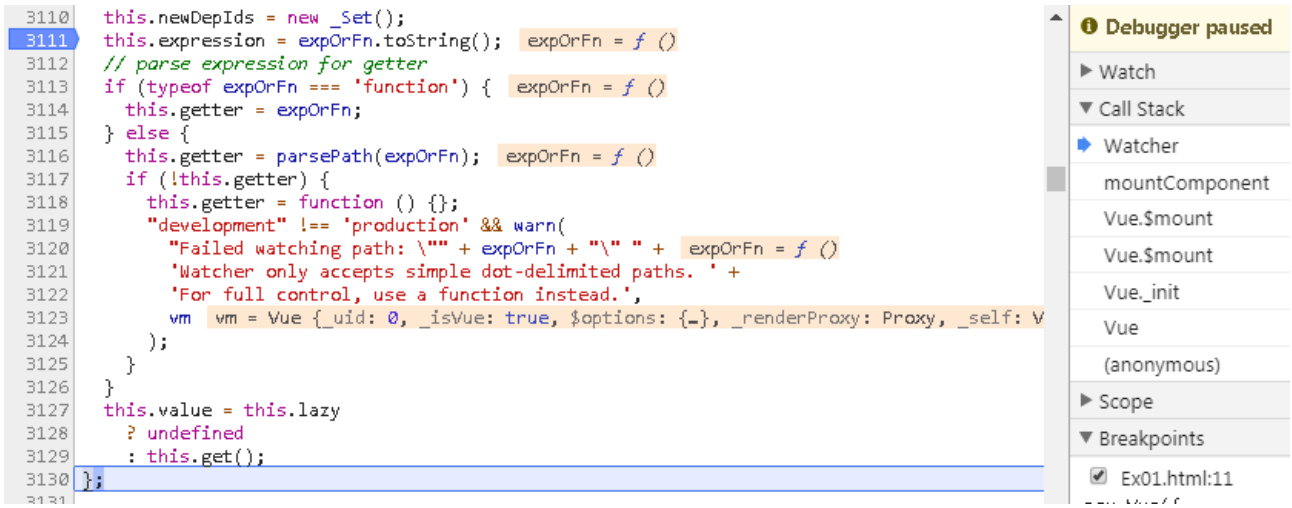
182. Retorno a updateComponent:



183. Retorno ao método ***get*** do objeto **Watcher**:

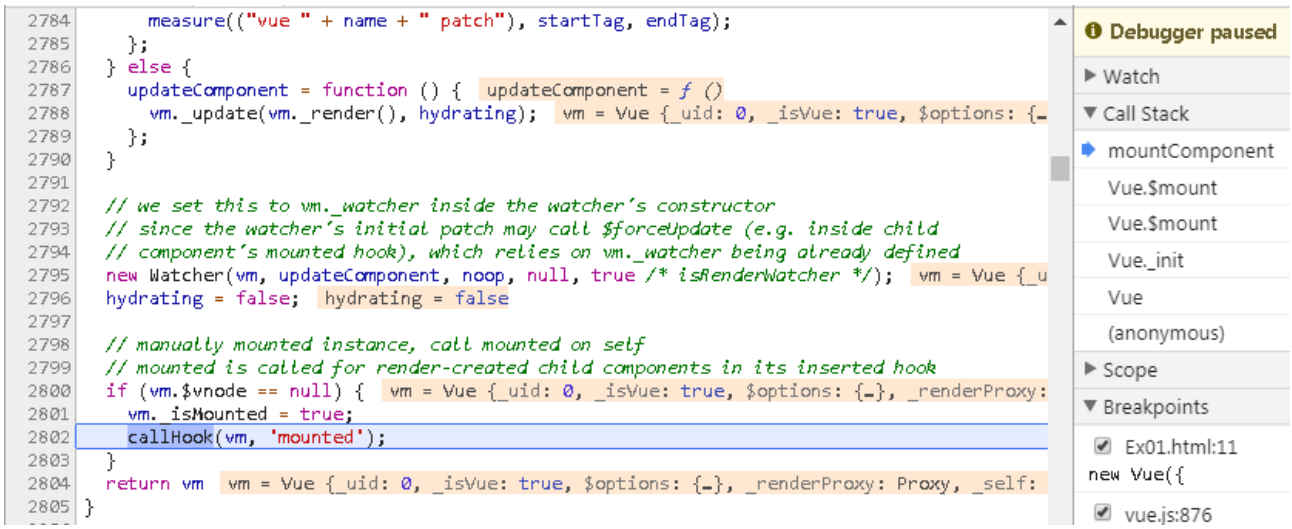


184. Retorno à **Watcher**:

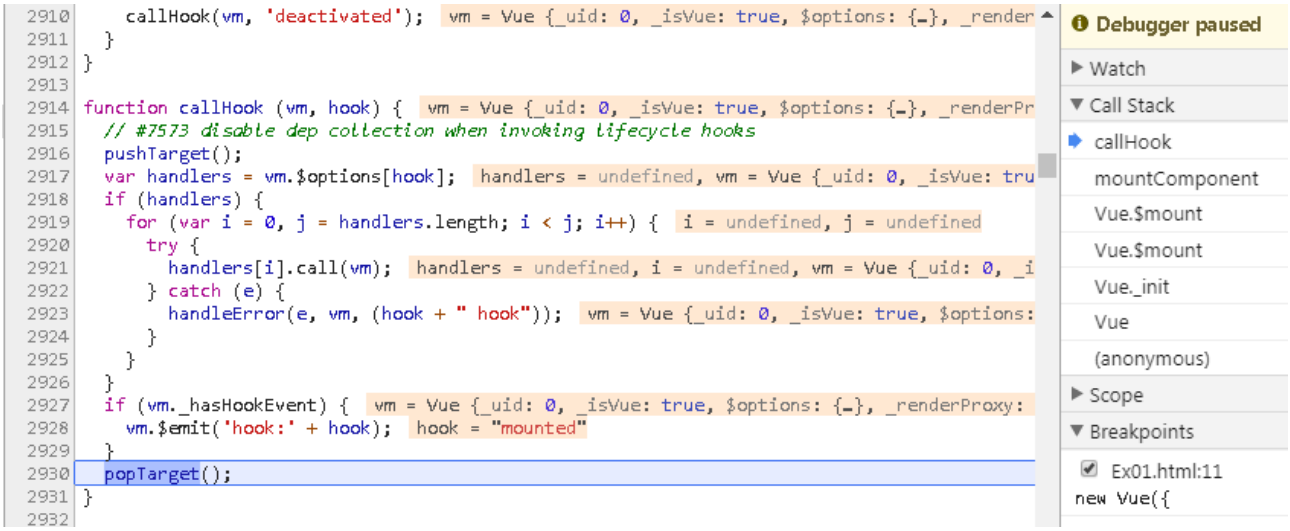


VUE – Substituição de variáveis

185. Retorno à ***mountComponent*** e chamada ao hook de “mounted”:

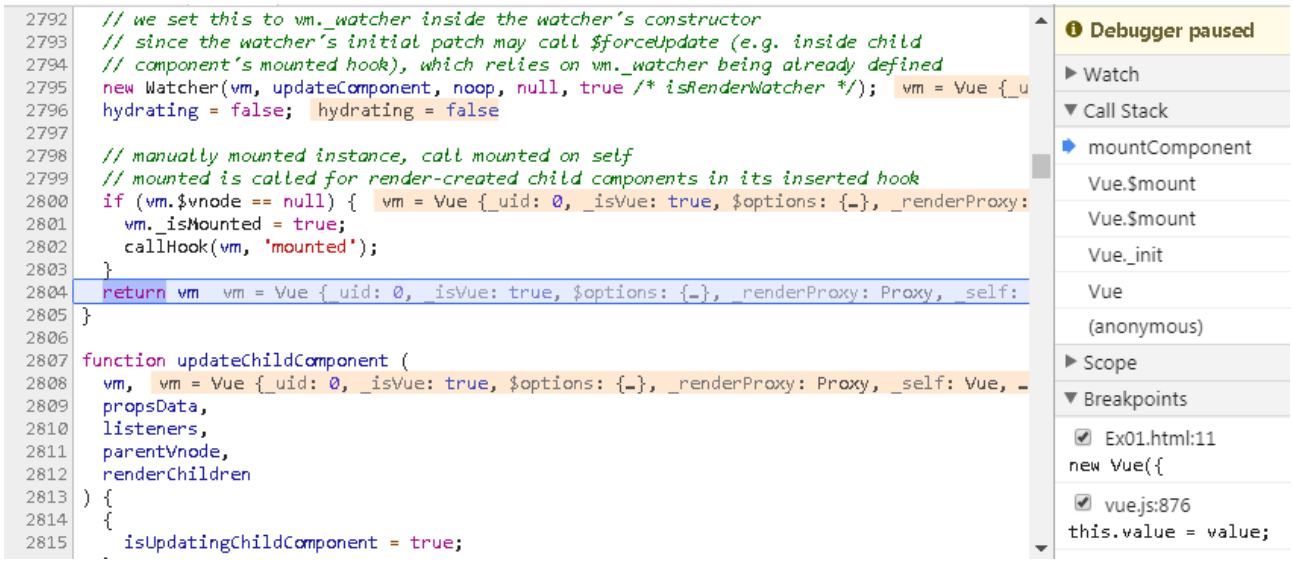


186. O *hook* ***mounted*** definido pelo usuário é chamado:



VUE – Substituição de variáveis

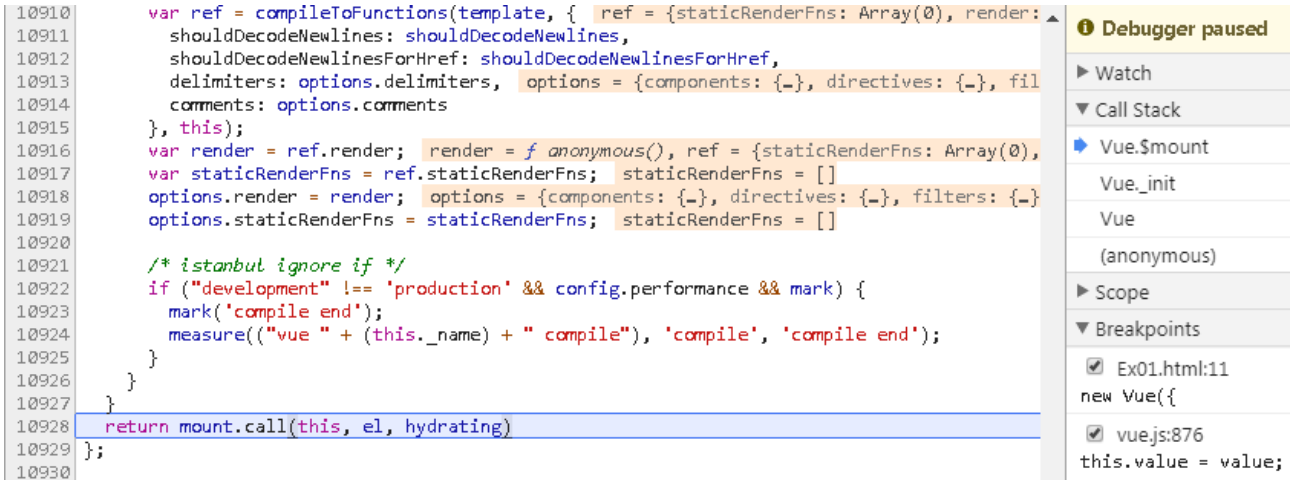
187. Preparando para retornar de ***mountComponent*** com o objeto **Vue** na instância **vm**:



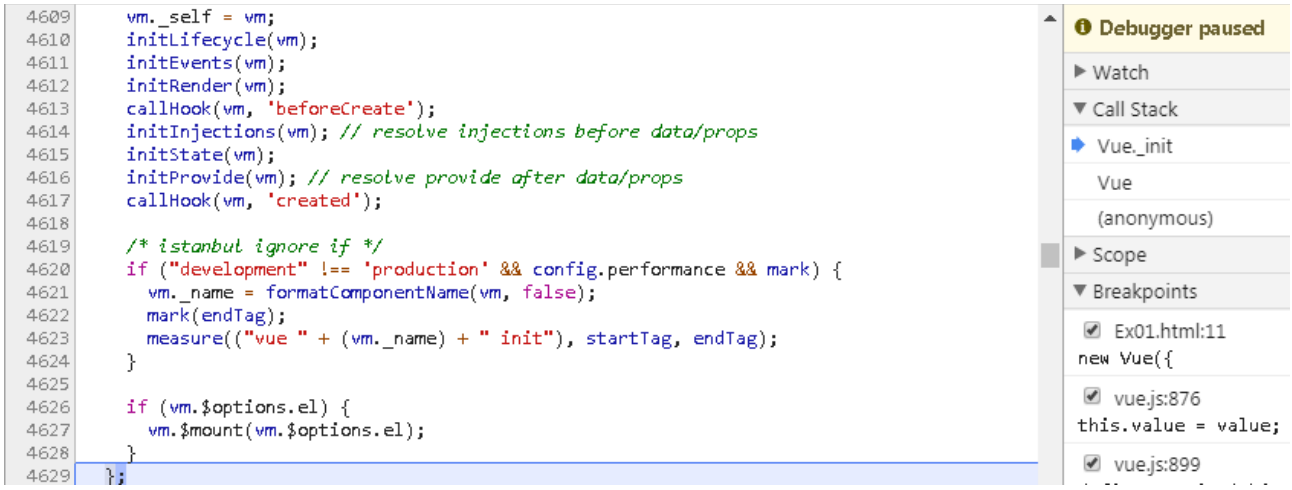
188. Retornando à ***$mount***:

VUE – Substituição de variáveis

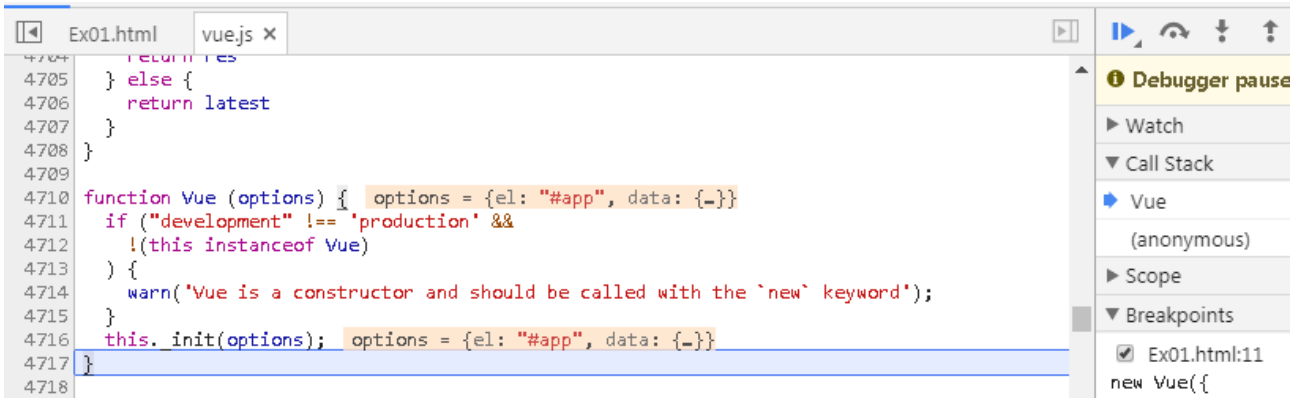
189. Retorno à chamada mount:



190. Retorno ***Vue._init***:



191. Retorno da definição da instância **Vue**:



VUE – Substituição de variáveis

192. Retorno ao código Javascript de nosso exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World - Vue.js 2</title>
    <script src="vue.js"></script>
</head>
<body>
    <div id="app">{{ author }} says: {{ message }}</div>
    <script type="text/javascript">
        new Vue({
            el: "#app",
            data: { author: 'Meyer',message: "Hello World!" }
        });
    </script>
</body>
</html>
```