为什么Vue2 this 能够直接获取到

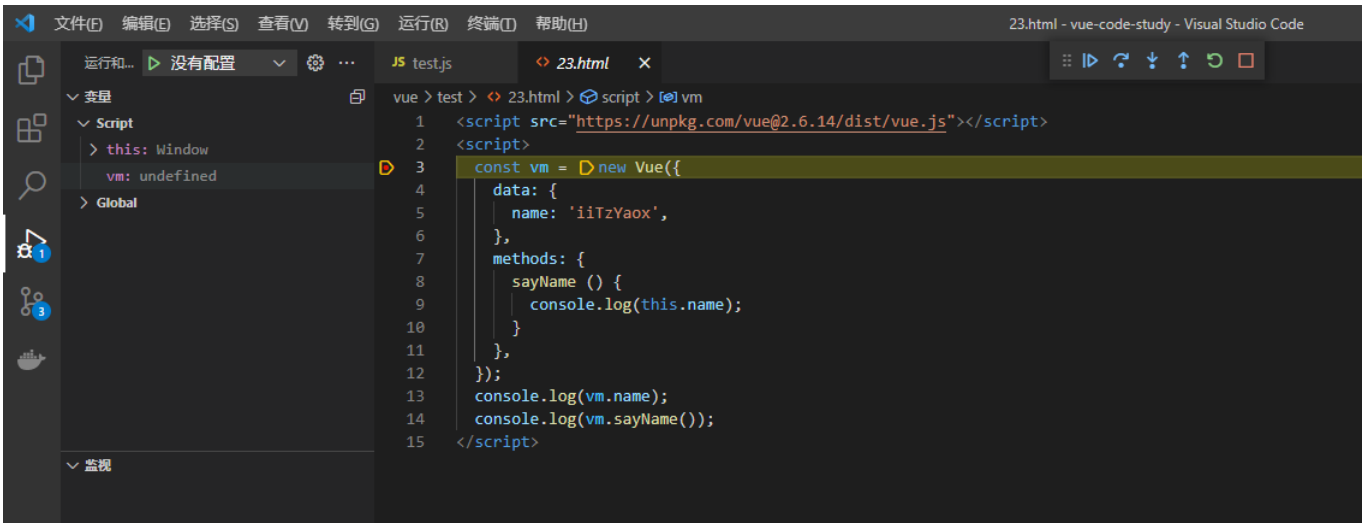
```
const vm = new Vue({
    data: {
        name: '我是若川',
    },
    methods: {
        sayName(){
            console.log(this.name);
        }
    },
});
console.log(vm.name); // 我是若川
console.log(vm.sayName()); // 我是若川
```

先贴上川哥的示例代码，为什么this能够直接访问到data里面的数据，为什么能获取到method方法呢?

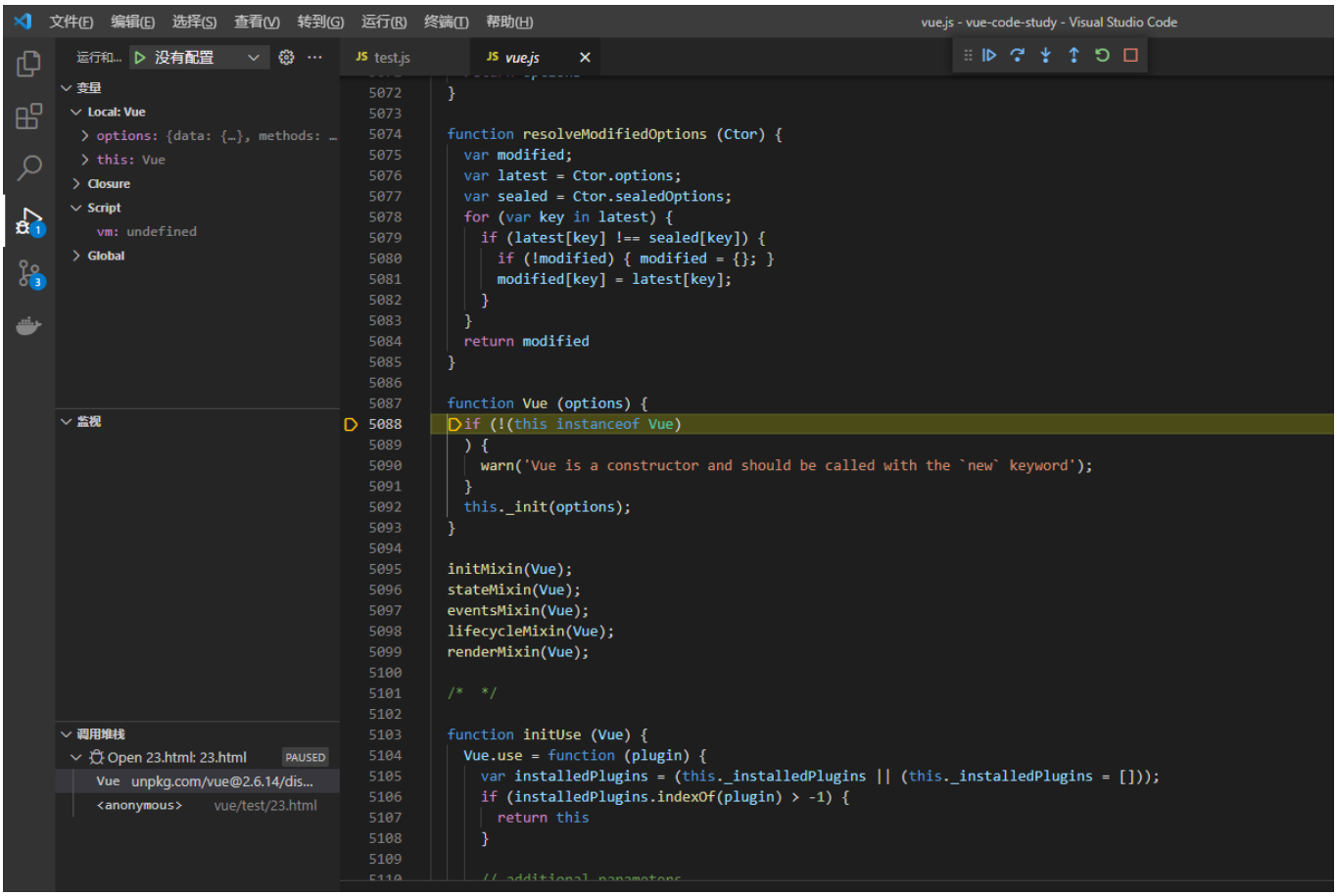
为了解决这些疑问，我们就去源码一探究竟吧。
首先创建一个html文件，代码如下

```
<script src="https://unpkg.com/vue@2.6.14/dist/vue.js"></script>
<script>
  const vm = new Vue({
    data: {
      name: 'iiTzYaox',
    },
    methods: {
      sayName () {
        console.log(this.name);
      }
    },
  });
  console.log(vm.name);
  console.log(vm.sayName());
</script>
```

川哥是在source中调试，我就直接在VSCode中调试了，直接在const vm = new Vue({ 这行打上断点，直接F5跑调试。



F11单步调试，此时我们就可以进入vue的构造函数啦



if (!(this instanceof Vue)

instanceof 运算符 object instanceof constructor

用于检测构造函数的 prototype属性是否出现在某个实例对象的原型链上。

```
instanceof (A,B) = {
    var L = A.__proto__;
    var R = B.prototype;
    if(L === R) {
        //A的内部属性__proto__指向B的原型对象
        return true;
    }
    return false;
}
```

如果不是new出来的（应该是指bind，call，apply之类的情况）就警告，如果是则init

继续F11，进入_init

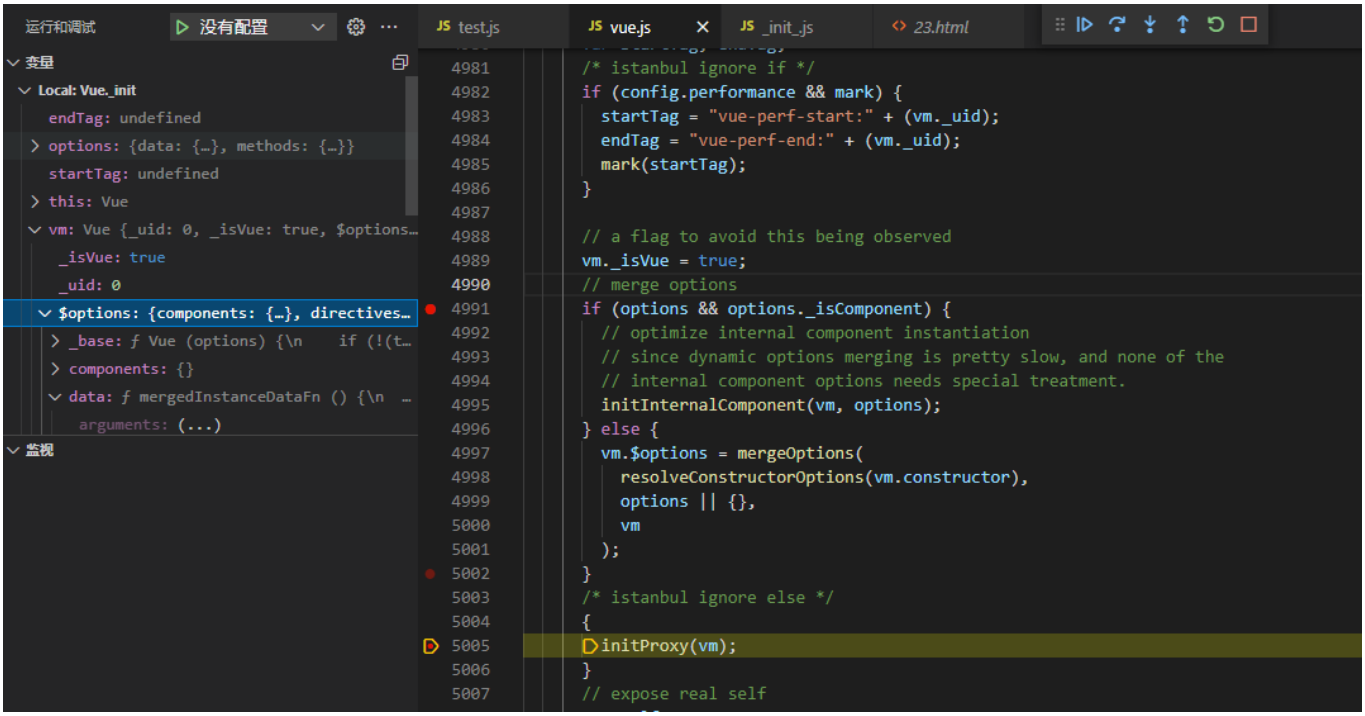
```
function initMixin (Vue) {
   Vue.prototype._init = function (options) {
     var vm = this;
     // a uid
     vm._uid = uid$3++;

     // vue performance的两个Tag,分析性能用，跟本次源码阅读无关
     var startTag, endTag;
     /* istanbul ignore if */
     if (config.performance && mark) {
       startTag = "vue-perf-start:" + (vm._uid);
       endTag = "vue-perf-end:" + (vm._uid);
       mark(startTag);
     }

     // a flag to avoid this being observed
     // 看着应该是把option丢入vm中，跟本次源码阅读无关
     vm._isVue = true;
     // merge options
     if (options && options._isComponent) {
       // optimize internal component instantiation
       // since dynamic options merging is pretty slow, and none of the
       // internal component options needs special treatment.
       initInternalComponent(vm, options);
     } else {
       vm.$options = mergeOptions(
         resolveConstructorOptions(vm.constructor),
         options || {},
         vm
       );
     }
     /* istanbul ignore else */
     {
       initProxy(vm);
     }
     // expose real
     // 暴露自己
     vm._self = vm;
     // 初始化生命周期
     initLifecycle(vm);
     // 初始化事件
     initEvents(vm);
     // 初始化渲染
     initRender(vm);
     callHook(vm, 'beforeCreate');
     initInjections(vm); // resolve injections before data/props
     initState(vm);
     initProvide(vm); // resolve provide after data/props
     callHook(vm, 'created');

     /* istanbul ignore if */
     if (config.performance && mark) {
       vm._name = formatComponentName(vm, false);
       mark(endTag);
       measure(("vue " + (vm._name) + " init"), startTag, endTag);
     }
```

```
    if (vm.$options.el) {
      vm.$mount(vm.$options.el);
    }
  };
}
```



options是我们html中new vue传入的对象
我们能够看到在init之前vm都没有能够实现直接this获取method，data
我们继续一些列的操作看着应该都跟我们的问题没有关系，一直到了initState

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

看代码应该是做了如下操作：
初始化props
初始化method
如果有data，初始化data，否则监视data
如果有computed，初始化computed
如果有watch，并且watch!=nativeWatch,初始化watch
感觉有戏，initMethod打断点,F5到断点处，F11进入initMethod

```
  function initMethods (vm, methods) {
    var props = vm.$options.props;
    for (var key in methods) {
      {
        // 判断method中每一项是否是函数，如果不是则警告
        if (typeof methods[key] !== 'function') {
          warn(
            "Method \"" + key + "\" has type \"" + (typeof methods[key]) + "\" in the component
definition. " +
            "Did you reference the function correctly?",
            vm
          );
        }
        // 判断methods中每一项是否与props冲突，如果冲突，警告
        if (props && hasOwn(props, key)) {
          warn(
            ("Method \"" + key + "\" has already been defined as a prop."),
            vm
          );
        }
        // 判断methods中每一项 是否已经在vm上，且方法是保留的，如果是则警告
        if ((key in vm) && isReserved(key)) {
          warn(
            "Method \"" + key + "\" conflicts with an existing Vue instance method. " +
            "Avoid defining component methods that start with _ or $."
          );
        }
      }
      // bind绑定到vm上!破案
      vm[key] = typeof methods[key] !== 'function' ? noop : bind(methods[key], vm);
    }
  }
```

抛开一系列的警告，我们发现最后一行
vm[key] = typeof methods[key] !== 'function' ? noop : bind(methods[key], vm);
破案成功!
这边bind之前vue/shared/utils中有过就不介绍了

下一步去initData看看

```
  function initData (vm) {
    var data = vm.$options.data;
    // 如果data是函数则把getData(data,vm)赋值给vm._data，不然就直接给data
    data = vm._data = typeof data === 'function'
      ? getData(data, vm)
      : data || {};
    // 如果不是纯对象，警告
    if (!isPlainObject(data)) {
      data = {};
      warn(
        'data functions should return an object:\n' +
        'https://vuejs.org/v2/guide/components.html#data-Must-Be-a-Function',
        vm
      );
    }
    // proxy data on instance
    var keys = Object.keys(data);
```

```
  var props = vm.$options.props;
  var methods = vm.$options.methods;
  var i = keys.length;
  // 与initMethods一样的操作，判断是否冲突如果冲突，警报
  while (i--) {
    var key = keys[i];
    {
      if (methods && hasOwn(methods, key)) {
        warn(
          ("Method \"" + key + "\" has already been defined as a data property."),
          vm
        );
      }
    }
    if (props && hasOwn(props, key)) {
      warn(
        "The data property \"" + key + "\" is already declared as a prop. " +
        "Use prop default value instead.",
        vm
      );
    } else if (!isReserved(key)) {
      proxy(vm, "_data", key);
      // 如果是保留属性，做代理，代理到_data上
    }
  }
  // observe data
  // 监测数据
  observe(data, true /* asRootData */);
}

// 如果是函数，则调用函数，获取对象。感觉就是处理闭包
function getData (data, vm) {
  // #7573 disable dep collection when invoking data getters
  pushTarget();
  try {
    return data.call(vm, vm)
  } catch (e) {
    handleError(e, vm, "data()");
    return {}
  } finally {
    popTarget();
  }
}


// 代理 之前传参是  proxy(vm, "_data", key);
// 把vm.xxx 代理到 vm._data.xxx
// 其实就是Object.defineProperty 定义对象
// 在vm上，定义属性vm.xxx是读写都是在vm._data.xxx上

function proxy (target, sourceKey, key) {
  sharedPropertyDefinition.get = function proxyGetter () {
    return this[sourceKey][key]
  };
  sharedPropertyDefinition.set = function proxySetter (val) {
    this[sourceKey][key] = val;
  };
  Object.defineProperty(target, key, sharedPropertyDefinition);
}
```

一开始看到initMethod以为data应该也是直接赋值，没想到竟然是通过defineProperty，来实现代理

总结：

Object.defineProperty，数据之间竟然也能proxy
反思：为什么自己从来都没有对this指向问题产生过疑问，应该对于技术持有一种好奇心。

TODO:手写一个简版的改变this指向的构造函数