问题1: v-for loop arr.shift() 时 进入死循环

问题描述:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>


  <div id='app'>
    <ul>
      <li v-for="(item, index) in data_nodes.shift()">{{ item }}</li>
      <li v-for="(item, index) in computed_nodes.shift()">{{ item }}</li>
    </ul>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
</head>

<body>
  <!--  -->
  <script>
    const app = new Vue({
      el: '#app',
      data: {
        data_nodes: [{}],
      },
      computed: {
        computed_nodes() {
          return [{
            content: '空置节点',
          }]
        }
      }
    });
  </script>

</body>
<html>
```

```
▶[Vue warn]: You may have an infinite update loop in a component render function.                    vue.js:634

(found in <Root>)
```

原因:

> vue底层重写了shift/unshift/pop/push/splice/sort/reverse方法, 实现了深层双向绑定; 因此用这些方法时, 会触发setter;
>
> data_nodes作为data选项, 当data_nodes的setter触发时, 通知watcher, 从而使他关联的组件重新渲染;

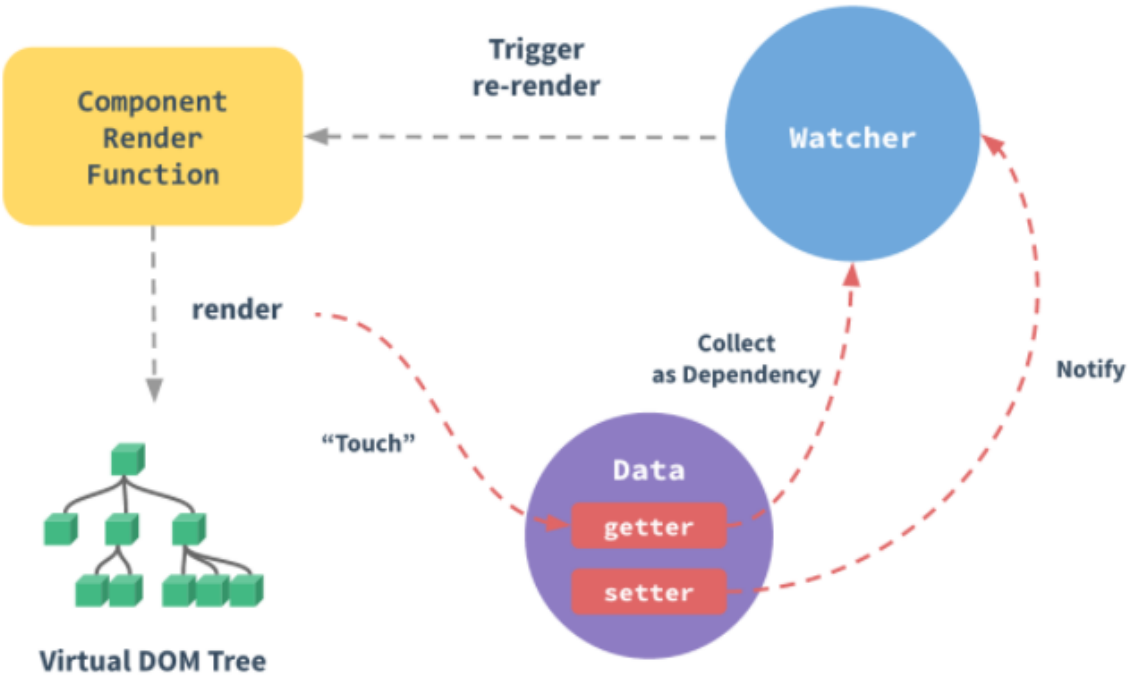
如何追踪变化

> 在 Vue Mastery 观看视频讲解

当你把一个普通的 JavaScript 对象传入 Vue 实例作为 `data` 选项，Vue 将遍历此对象所有的 property，并使用 `Object.defineProperty` 把这些 property 全部转为 getter/setter。`Object.defineProperty` 是 ES5 中一个无法 shim 的特性，这也就是 Vue 不支持 IE8 以及更低版本浏览器的原因。

这些 getter/setter 对用户来说是不可见的，但是在内部它们让 Vue 能够追踪依赖，在 property 被访问和修改时通知变更。这里需要注意的是不同浏览器在控制台打印数据对象时对 getter/setter 的格式化并不同，所以建议安装 vue-devtools 来获取对检查数据更加友好的用户界面。

每个组件实例都对应一个 **watcher** 实例，它会在组件渲染的过程中把“接触”过的数据 property 记录为依赖。之后当依赖项的 setter 触发时，会通知 watcher，从而使它关联的组件重新渲染。



```
var methodsToPatch = [
  'push',
  'pop',
  'shift',
  'unshift',
  'splice',
  'sort',
  'reverse'
];
```

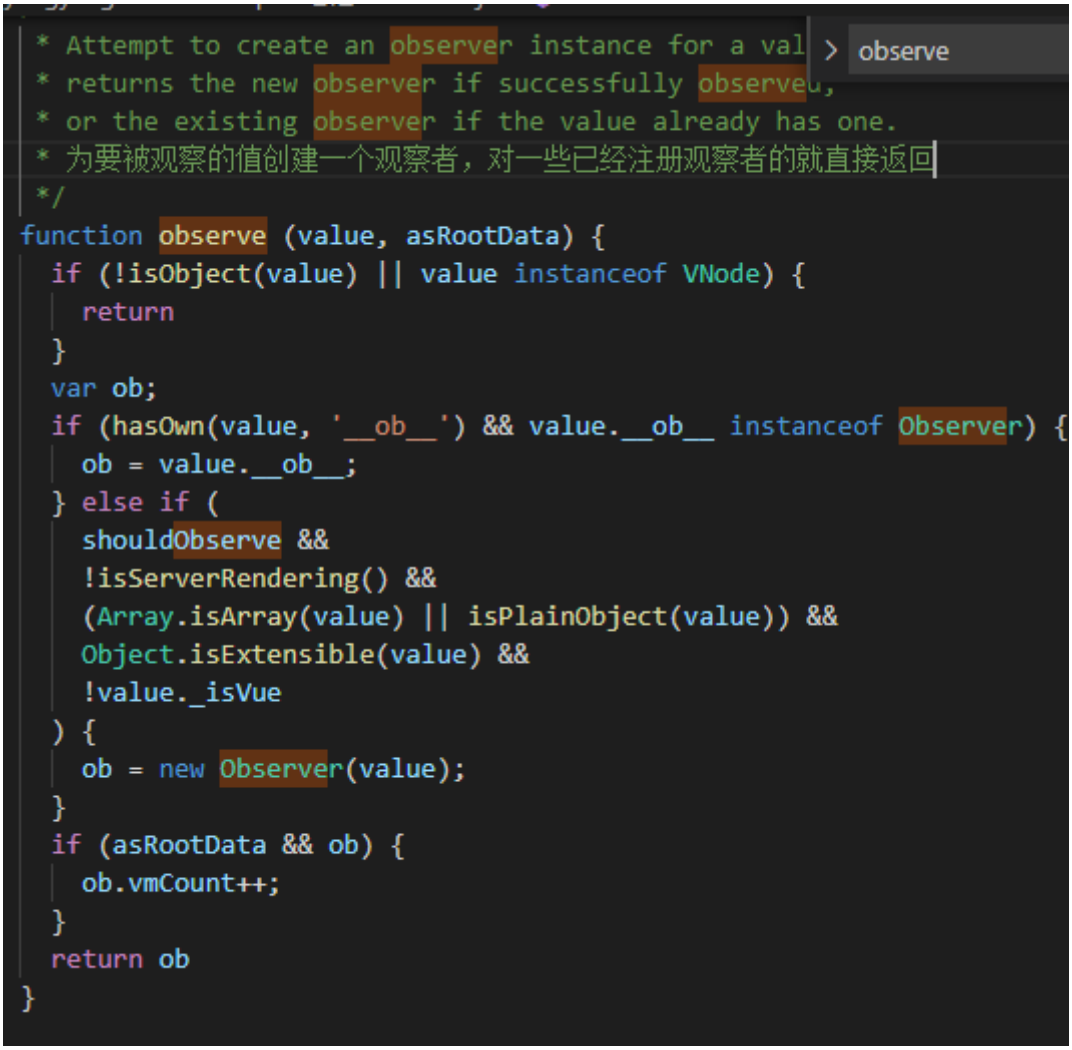
```
methodsToPatch.forEach(function (method) {
  // cache original method
  var original = arrayProto[method];
  def(arrayMethods, method, function mutator () {
    var args = [], len = arguments.length;
    while ( len-- ) args[ len ] = arguments[ len ];

    var result = original.apply(this, args);
    var ob = this.__ob__;
    var inserted;
    switch (method) {
      case 'push':
      case 'unshift':
        inserted = args;
        break
      case 'splice':
        inserted = args.slice(2);
        break
    }
    if (inserted) { ob.observeArray(inserted); }
    // notify change
    ob.dep.notify();
    return result
  });
});
```

```
/**
 * Observe a list of Array items.
 */
Observer.prototype.observeArray = function observeArray (items) {
  for (var i = 0, l = items.length; i < l; i++) {
    observe(items[i]);
  }
};
```

> [__ ob__ : Observer]这些数据是vue这个框架对数据设置的监控器，一般都是不可枚举的。

```
// 检查被监听的值是否已经被监听过了，就是通过是否有__ob__属性并且__ob__属性是否是Observe的实例对象来判断的
```



> 总结:
>
> data, props里的数据, setter触发时会通知watcher, 关联组件重新渲染;
>
> v-for loop的变量不要触发setter
>
> 支持响应式更新的：push，pop, shift, unshift, splice，sort, reverse 不支持响应式更新的：filter, concat, slice

> 参考:
>
> Vue父子组件生命周期执行顺序

问题2:Qianxin Design <q-steps>组件, index乱序

问题描述:

```
      <q-steps :space="200" :active="activeNodeIndex">
        <q-step v-for="(item) in scheduleNodes" :key="item.id"
:title="item.name">
          <div slot="description">
            <p>1号点{{item}}</p>
            <p>2号点{{item}}</p>
          </div>
        </q-step>
        <q-step>3号点</q-step>
      </q-steps>
```

预期效果:

实际效果:

原因:原来分析:父子组件的create都在父子组件的mounted之前, 子组件渲染时, 最后的节点最先渲染,被push进steps[], v-for的节点后续获取;

后来发现这个原因不对,push顺序还是安装v-for先,div后,因为它们都写在templete内;真正原因是因为便利的数据是异步的,而div的数据是同步的;

> https://codepen.io/weimengxi/pen/xxdwyZL

父组件: steps.vue:

```
data() {
   return {
     steps: [], // step 组件创立之前将组件实例增加到父组件的 steps 数组中
     stepOffset: 0
   };
 },

watch: {
   active(newVal, oldVal) {
     this.$emit('change', newVal, oldVal);
   },
   steps(steps) {
     steps.forEach((child, index) => {
       child.index = index; // 设置子组件的 index 属性，将会用于子组件的展现逻辑
     });
   }
 }
```

子組件: step.vue

```
// step 组件创立之前将组件实例增加到父组件的 steps 数组中
beforeCreate() {
  this.$parent.steps.push(this);
},

data() {
  return {
    index: -1,
  };
},

methods: {
  getCurrentStatus() {
    // 拜访父组件的逻辑更新属性
    const { current, type, steps, timeForward } = this.$parent;
    // 逻辑解决
  }
},
mounted() {
  // 监听 index 的变动从新计算相干逻辑
  const unwatch = this.$watch('index', val => {
    this.$watch('$parent.current', this.getCurrentStatus, { immediate: true });
    unwatch();
  });
}
```

最终实现方式:

```
7   8         <q-steps :space="200" :active="activeNodeIndex">
8       -       <q-step v-for="(item) in nodes" :key="item.id" :title="item.name">
    9   +       <q-step v-for="(item) in scheduleNodes" :key="item.id" :title="item.name">
9   10            <div slot="description">
10      -           <p>{{ item.type === 'NodeType_START' ? '创建人' : '审批人' }}: {{ item.roleNames }}</p>
11      -           <p>{{ item.type === 'NodeType_START' ? item.createTime : item.approvalTime }}</p>
    11  +           <p v-if="item.type === &quot;NodeType_START&quot;">
    12  +             {{ '创建人' }}: {{ item.roleNames }}
    13  +           </p>
    14  +           <p v-if="item.type === &quot;NodeType_END&quot;">
    15  +             {{ '审批人' }}: {{ item.roleNames }}
    16  +           </p>
    17  +           <p v-if="item.type === &quot;NodeType_COMPLETED&quot;" />
12  18          </div>
13  19        </q-step>
14  20      </q-steps>
```

> 总结: 同级并发的时候, 不要一起用v-for和普通的组件, 可能会乱序
>
> 源码改进方式: 参考 provide+inject+created

父子组件通信时的生命周期测试: