VUE数据响应系统与组件渲染函数

参照：example_1.html

html：

```
<div id="app">
</div>
```

Javascript：

```
var count = 0;

var app = new Vue({
  el: '#app',

  render: function (createElement) {
    console.log('render')
    count++
    return Vue.compile(`
      <div>
        <p>渲染次数：${count}次</p>
        <p>{{ info.message }}</p>
      </div>
    `).render.call(this,createElement)
  },

  data: {
    message: 'Hello Vue!',
    info: {
      message: 'Hello Vue!',
      code: '200'
    }
  }
});
```

不考虑依赖收集的详细过程，有以下结论：

1.渲染函数，对info本身和info的message属性存在依赖关系。
2.渲染函数，对message及info的code属性不存在依赖关系。

以下讨论，响应式数据对象发生变化。依赖触发的渲染函数情况

对应的响应式对象app.$data

```
info:Object
message:"Hello Vue!"
__ob__:Observer {value: {…}, dep: Dep, vmCount: 1}
get info:ƒ reactiveGetter()
set info:ƒ reactiveSetter(newVal)
get message:ƒ reactiveGetter()
set message:ƒ reactiveSetter(newVal)
```

对应的响应式对象app.$data.info

```
code:"200"
message:"Hello Vue!"
__ob__:Observer {value: {…}, dep: Dep, vmCount: 0}
get code:ƒ reactiveGetter()
set code:ƒ reactiveSetter(newVal)
get message:ƒ reactiveGetter()
set message:ƒ reactiveSetter(newVal)
```

Dep对象，Vue中用来管理依赖的对象，可以被多个Watcher订阅。
每一组geter/seter（reactiveGetter 和 reactiveSetter）都闭包引用了一个Dep实例对象。
例子：

```
app.info.message = 'Hello JavaScript!'
reactiveSetter  → 闭包：dep.notify() → vm._watcher.update() → render函数
```

__ob__.dep 也是一个 Dep对象。
__ob__ 属性以及 __ob__.dep 的主要作用是为了添加、删除属性时有能力触发依赖，而这就是 Vue.set 或 Vue.delete 的原理。
使用Vue.set更新已存在属性时，这不会触发__ob__.dep。
如果响应式对象是一个Array，响应式对象的arrayMethods。也会触发依赖

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

```
];
```

例子：

```
Vue.set(app.info, 'error', 'no error')
Vue.set → op.dep.notify() → vm._watcher.update() → render函数
```

与渲染函数存在依赖关系的Dep对象实例

app.$data.__ob__.dep	由于app是个Root组件，无法使用vue.set触发。
app.$data.info的geter/seter闭包引用的dep	对info的值进行更新即可触发。如 info = {};
app.$data.info.__ob__.dep	使用Vue.set对info进行添加或删除属性时。如Vue.set(app.info, 'error', 'no error')。
app.$data.info.message的geter/seter闭包引用的dep	对info.message的值进行更新。如app.info.message = 'Hello JavaScript!' 。

参照：example_2.html

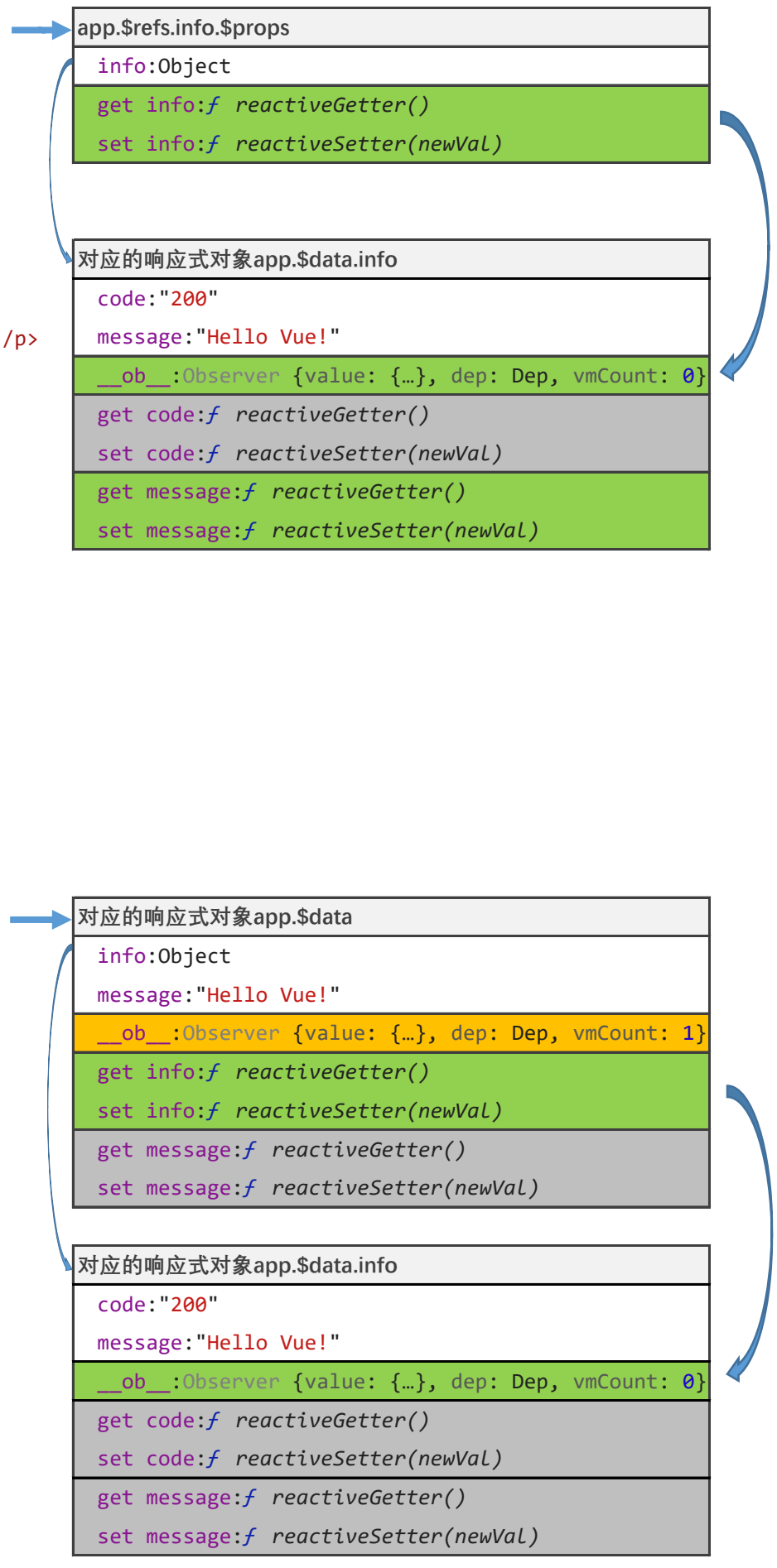
```
var count = 0;
var countOfInfo = 0;

Vue.component('info', {
  render: function (createElement) {

    console.log('info render')
    countOfInfo++

    return Vue.compile(`
      <div>
        <p>{{info.message}} 渲染次数：${countOfInfo}次</p>
      </div>
    `).render.call(this, createElement)
  },
  data () {
    return {}
  },
  props: {
    info: {
      type: Object,
      require: true
    }
  }
})

var app = new Vue({
  el: '#app',
  render: function (createElement) {
    console.log('render')
    count++
    return Vue.compile(`
      <div>
        <p>渲染次数：${count}次</p>
        <info :info="info" ref="info"></info>
      </div>
    `).render.call(this,createElement)
  },
  data: {
    message: 'Hello Vue!',
    info: {
      message: 'Hello Vue!',
      code: '200'
    }
  }
});
```



参照：example_3.html

```
var count = 0;

var app = new Vue({
  el: '#app',
  render: function (createElement) {
    console.log('render')
    count++
    return Vue.compile(`
      <div>
        <p>渲染次数：${count}次</p>
        <p>0: {{ infos[0].message }}</p>
        <p>1: {{ infos[1].message }}</p>
      </div>
    `).render.call(this,createElement)
  },
  data: {
    infos: [{
      message: 'Hello Vue!'
    },{
      message: 'Hello Vue!!!'
    }]
  }
});
```