《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

第20讲：详解组件通信之状态管理

组件

拉勾教育
互联网人实战大学

数据模型、渲染和视图

通过组件化的方式能够有效地将 Web 页面进行解耦

组件之间如何进行通信 ？

全局状态

父子组件通信：

父组件通过 prop(s) 属性向子组件传参

子组件通过自定义事件来向父组件发送消息

非父子组件之间如果通过层层传递

这个过程就会变得相当麻烦

全局状态

直接使用全局变量会存在一些问题

- 可能多个组件会同时修改变量值

  这个过程无法追踪，调试问题也会变得很麻烦

- 当全局变量值发生变化时

  如何通知引用它的每一个组件？

状态管理库的特点

“状态”就是不同组件之间传递和引用的数据模型

状态管理库具有 3 个特点：**可预测、中心化、可调式**

状态管理库的特点

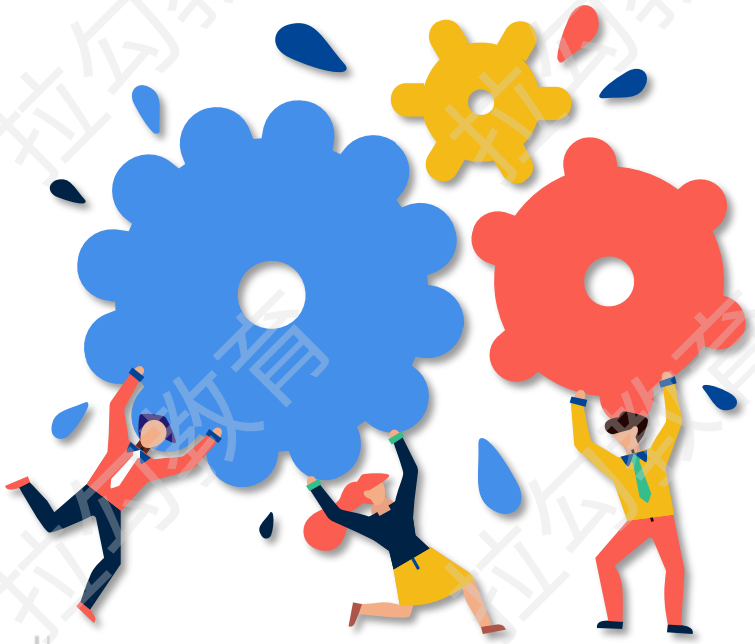
可预测

如果状态 A 经过操作 B 会生成状态 C

那么不论在任何时刻、任何平台（客户端、服务端、App 端）

只要 A 和 B 不发生变化

就能得到同样的结果 C



状态管理库的特点


```
function getTime() {
  return new Date().getTime()
}

function getDom(id) {
  return document.getElementById(id)
}
```

状态管理库的特点

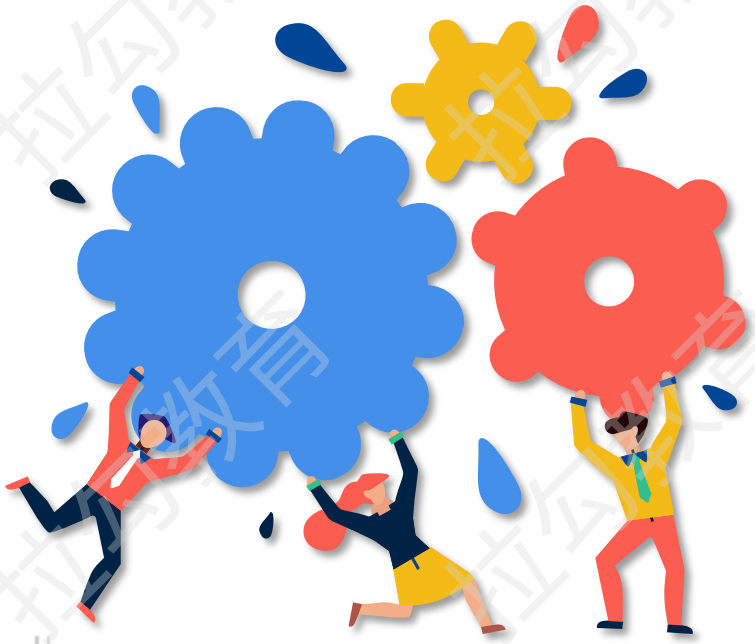

```
function nextDay(time) {
  return new Date(time + 1000 * 60 * 60 * 24)
}
function filter(a, b){
  return a + b
}
```

状态管理库的特点

中心化

Vuex 和 Redux 都只会构建一棵中心化的状态树

所有的状态数据都会作为子属性挂载到这棵树上



状态管理库的特点

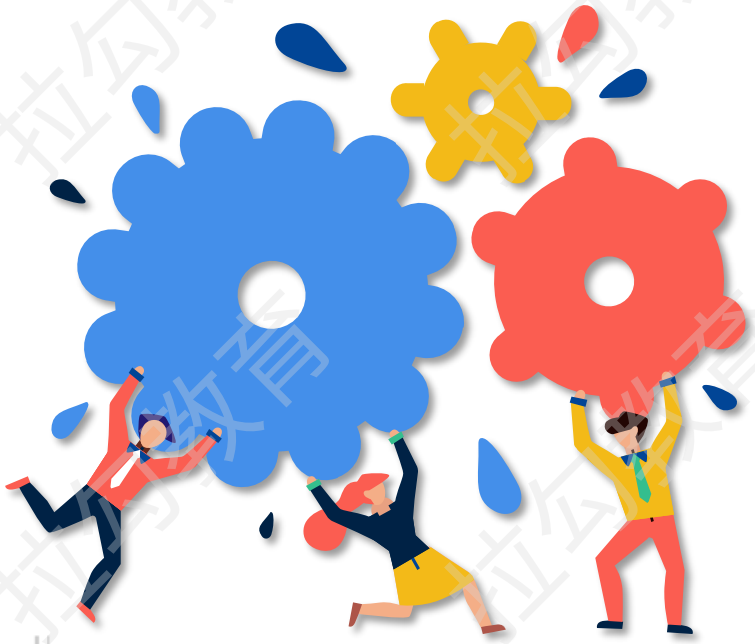
可调式

可调式指的是可以利用**浏览器插件**

对状态的变化和使用情况进行追踪和调试

Vuex 提供了 Vue.js devtools 插件

Redux 也提供了 Redux DevTools



状态管理库实现原理


```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state, payload) {
      state.count += payload
    }
  }
})
store.commit('increment', 1)
console.log(store.state.count) // -> 1
```

状态管理库实现原理


```
Store.prototype.commit = function commit (_type, _payload, _options) {
 var this$1 = this;
 ...
 var mutation = { type: type, payload: payload };
 var entry = this._mutations[type];
 ...
 this._withCommit(function () {
  entry.forEach(function commitIterator (handler) {
   handler(payload);
  });
 });
```

状态管理库实现原理


```
  });
});
...
};

Store.prototype._withCommit = function _withCommit (fn) {
  var committing = this._committing;
  this._committing = true;
  fn();
  this._committing = committing;
};
```

状态管理库实现原理


```
store._vm = new Vue({
  data: {
    $$state: state
  },
  computed: computed
});
```

状态管理库实现原理


```
Object.defineProperties( Store.prototype,
prototypeAccessors$1 );
prototypeAccessors$1.state.get = function () {
 return this._vm._data.$$state
};
```

状态管理库实现原理


```
function counter(state = 0, action) {
  switch (action.type){
    case 'INCREMENT':
      return state + 1
    case 'DECREMENT':
      return state - 1
    default:
      return state
  }
}
```

状态管理库实现原理


```
}

let store = createStore(counter)

store.subscribe(() => console.log(store.getState()))

store.dispatch({ type: 'INCREMENT' })// 1
store.dispatch({ type: 'INCREMENT' })// 2
store.dispatch({ type: 'DECREMENT' })// 1
```

状态管理库实现原理


```
function dispatch(action) {
  ...
  try {
    isDispatching = true;
    currentState = currentReducer(currentState, action);
  } finally {
    isDispatching = false;
  }
  ...
  return action;
}
```

状态管理库实现原理


```
function getState() {
  if (isDispatching) {
    throw new Error('You may not call store.getState() while the reducer is executing. ' + 'The reducer has already received the state as an argument. ' + 'Pass it down from the top reducer instead of reading it from the store.');
  }

  return currentState;
}
```

其他组件通信方式

1.全局上下文

Vue 提供了一组 API 用来解决祖先组件与子孙组件的通信问题

那就是 **provide** 和 **inject**

provide 可以在祖先组件中指定我们想要提供给子孙组件的数据或方法

而在任何子孙组件中

我们都可以使用 **inject** 来接收 provide 提供的数据或方法

其他组件通信方式


```
<div id="app">
  <button v-on:click="o.count++">{{o.count}}</button>
  <button-counter></button-counter>
  <button-counter></button-counter>
  <button-counter></button-counter>
</div>

<script>
Vue.component('button-counter', {
  inject: ['o'],
  methods:{
    click() {
```

其他组件通信方式


```
    this.o.count++
   }
  },
  template: '<button v-on:click="click">You clicked me
{{ o.count }} times.</button>'
})
const app = new Vue({
  el: '#app',
  data: {
   o: {
    count: 0
   }
```

其他组件通信方式


```
data:{
  o:{
    count: 0
  }
},
provide(){
  return {
    o: this.o
  }
}
})
</script>
```

其他组件通信方式


```
data:{
  o:{
    count: 0
  }
},
provide(){
  return {
    o: this.o
  }
}
})
</script>
```

其他组件通信方式

2.事件监听

事件监听则是利用组件库本身的事件机制

设置一个**全局事件代理**

用来负责向各个组件传递数据

其他组件通信方式

拉勾教育
互联网人实战大学

```
<div id="app">
 <button v-on:click="click()">{{this.count}}</button>
 <button-counter></button-counter>
 <button-counter></button-counter>
 <button-counter></button-counter>
</div>
<script>
 var EventBus = new Vue();
 Object.defineProperties(Vue.prototype, {
  $bus: {
    get: function () {
```

其他组件通信方式


```
    return EventBus
  }
 }
})
Vue.component('button-counter', {
 mounted(){
  this.$bus.$on('count', c => this.count = c)
 },
 data() {
  return {
   count: 0
```

其他组件通信方式


```
    }
  },
  methods: {
    click() {
      this.$bus.$emit('count', this.count + 1)
    }
  },
  template: '<button v-on:click="click">You clicked me
{{ this.count }} times.</button>'
})
const app = new Vue({
```

其他组件通信方式


```
const app = new Vue({
 el: '#app',
 data: {
  count: 0
 },
 mounted() {
  this.$bus.$on('count', c => this.count = c)
 },
 methods: {
  click() {
   this.$bus.$emit('count', this.count + 1)
```

其他组件通信方式


```
  },
  mounted() {
    this.$bus.$on('count', c => this.count = c)
  },
  methods: {
    click() {
      this.$bus.$emit('count', this.count + 1)
    }
  }
})
</script>
```

总结

介绍了 3 种不同的**跨组件通信方式**

通信双方不属于父子组件，也就是没有直接的依赖/引用关系

所以需要借助“第三方”来进行传递数据

这些“第三方”既包括视图库（Vue 和 React）本身提供的事件机制

或全局上下文

也包括面向其进行开发的状态管理库

总结

介绍了 3 种不同的**跨组件通信方式**

深入分析全局状态管理库 Vuex 和 Redux 的源码

理解了其实现原理

总结

介绍了 3 种不同的**跨组件通信方式**

组件库默认提供了全局上下文的方式来解决跨组件通信问题

非常轻量，适合在小型 Web 应用中使用

缺点是追踪调试状态变化比较困难

总结

拉勾教育
互联网人实战大学

你还知道哪些跨组件通信的方式？

Next：第21讲：《你的代码到底是怎么编译的？》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U