程序设计原则

设计模式非常多，每解决一个问题都会形成设计模式。 随着系统的迭代，系统的设计模式也在迭代。

实际项目中，无法在第一时间将所有东西都用到位，更不能为了用模式而用模式——那么该怎么思考呢?

接下来我们讨论几个思考软件设计时的通用原则。

Rule of Three

不要出现3份类似的程序。

密封性和单一职责

为什么Antd中的 Select和Option是分开的？Tab和Panel是分开的?

```
const Option = Select.Option

const TabPanel = Tab.Panel
```

为什么作业1中的UILayerView应该先实现TreeView？

延伸下：关注点分离原则

出自：Composition API proposalhttps://github.com/vuejs/rfcs/blob/master/active-rfcs/0013-composition-api.md

file-explorer-before.js

Raw

```
import { isValidPathName } from '@/util/folders'
import FOLDER_CURRENT from '@/graphql/folder/folderCurrent.gql'
import FOLDERS_FAVORITE from '@/graphql/folder/foldersFavorite.gql'
import FOLDER_OPEN from '@/graphql/folder/folderOpen.gql'
import FOLDER_OPEN_PARENT from '@/graphql/folder/folderOpenParent.gql'
import FOLDER_SET_FAVORITE from '@/graphql/folder/folderSetFavorite.gql'
import PROJECT_CWD_RESET from '@/graphql/project/projectCwdReset.gql'
import FOLDER_CREATE from '@/graphql/folder/folderCreate.gql'
const SHOW_HIDDEN = 'vue-ui.show-hidden-folders'

export default {
  data () {
    return {
      loading: 0,
      error: false,
      editingPath: false,
      editedPath: '',
      folderCurrent: {},
      foldersFavorite: [],
      showHidden: localStorage.getItem(SHOW_HIDDEN) === 'true',
      showNewFolder: false,
      newFolderName: ''
    }
  },

  apollo: {
    folderCurrent: {
      query: FOLDER_CURRENT,
      fetchPolicy: 'network-only',
      loadingKey: 'loading',
      async result () {
        await this.$nextTick()
        this.$refs.folders.scrollTop = 0
      }
    },
    foldersFavorite: FOLDERS_FAVORITE
  },

  computed: {
    newFolderValid () {
      return isValidPathName(this.newFolderName)
    }
  },

  watch: {
    showHidden (value) {
      if (value) {
        localStorage.setItem(SHOW_HIDDEN, 'true')
      } else {
        localStorage.removeItem(SHOW_HIDDEN)
      }
    }
  },

  beforeRouteLeave (to, from, next) {
    if (to.matched.some(m => m.meta.needProject)) {
      this.resetProjectCwd()
    }
    next()
  },

  methods: {
    async openFolder (path) {
      this.editingPath = false
      this.error = null
      this.loading++
      try {
        await this.$apollo.mutate({
          mutation: FOLDER_OPEN,
          variables: {
            path
          },
          update: (store, { data: { folderOpen } }) => {
            store.writeQuery({ query: FOLDER_CURRENT, data: { folderCurrent: folderOpen } })
          }
        })
      } catch (e) {
        this.error = e
      }
      this.loading--
    },

    async openParentFolder (folder) {
      this.editingPath = false
      this.error = null
      this.loading++
      try {
        await this.$apollo.mutate({
          mutation: FOLDER_OPEN_PARENT,
          update: (store, { data: { folderOpenParent } }) => {
            store.writeQuery({ query: FOLDER_CURRENT, data: { folderCurrent: folderOpenParent } })
          }
        })
      } catch (e) {
        this.error = e
      }
      this.loading--
    },

    async toggleFavorite () {
      await this.$apollo.mutate({
        mutation: FOLDER_SET_FAVORITE,
        variables: {
          path: this.folderCurrent.path,
          favorite: !this.folderCurrent.favorite
        },
        update: (store, { data: { folderSetFavorite } }) => {
          store.writeQuery({ query: FOLDER_CURRENT, data: { folderCurrent: folderSetFavorite } })
          let data = store.readQuery({ query: FOLDERS_FAVORITE })
          // TODO this is a workaround
          // See: https://github.com/apollographql/apollo-client/issues/4831#issuecomment-423668473
          data = {
            foldersFavorite: data.foldersFavorite.slice()
          }
          if (folderSetFavorite.favorite) {
            data.foldersFavorite.push(folderSetFavorite)
          } else {
            const index = data.foldersFavorite.findIndex(
              f => f.path === folderSetFavorite.path
            )
            index !== -1 && data.foldersFavorite.splice(index, 1)
          }
          store.writeQuery({ query: FOLDERS_FAVORITE, data })
        }
      })
    },

    cwdChangedUpdate (previousResult, { subscriptionData }) {
      return {
        cwd: subscriptionData.data.cwd
      }
    },

    async openPathEdit () {
      this.editedPath = this.folderCurrent.path
      this.editingPath = true
      await this.$nextTick()
      this.$refs.pathInput.focus()
    },

    submitPathEdit () {
      this.openFolder(this.editedPath)
    },

    refreshFolder () {
      this.openFolder(this.folderCurrent.path)
    },

    resetProjectCwd () {
      this.$apollo.mutate({
        mutation: PROJECT_CWD_RESET
      })
    },

    slicePath (path) {
      const parts = []
      let startIndex = 0
      let index
      const findSeparator = () => {
        index = path.indexOf('/', startIndex)
        if (index === -1) index = path.indexOf('\\', startIndex)
        return index !== -1
      }
      const addPart = index => {
        const folder = path.substring(startIndex, index)
        const slice = path.substring(0, index + 1)
        parts.push({
          name: folder,
          path: slice
        })
      }
      while (findSeparator()) {
        addPart(index)
        startIndex = index + 1
      }
      if (startIndex < path.length) addPart(path.length)
      return parts
    },

    async createFolder () {
      if (!this.newFolderValid) return
      const result = await this.$apollo.mutate({
        mutation: FOLDER_CREATE,
        variables: {
          name: this.newFolderName
        }
      })
      this.openFolder(result.data.folderCreate.path)
      this.newFolderName = ''
      this.showNewFolder = false
    }
  }
}
```

Options API

Composition API

总结下：

- 每件事情应该有独立的模块处理
- 每个独立的模块要把事情做好、做完整

32个棋 1个棋盘

单向依赖原则

在单向数据流中讨论非常信息了。 组件不要发生双向依赖，如果发生双向依赖可以这样解耦：

- 消息（EventBus, Redux……）
- 重新设计(skedo早先版本对比)

SSOT原则

Single Source of Truth

数据的来源只有1个，真理只有一个。

关联的原则：最小知识原则。

```
const ProductList = (props) => {// {a,b,c,d}
    const [passProps, setProps] = useState(props)

    return <X {...passProps}>
    </X>
}
```

举例1：Restful

举例2：减少组件间参数传递

反模式的设计：商品表单 -> 品牌子表单 -> 品牌列表

反思： 组件从数据层面也是密封的(sealed)。例如一个订单列表组件应该自己可以完成所有数据的获取，即便为了提升性能数据作为一个整体被服务端返回。

最小交互原则

减少类型间的交互，减少类型之间的耦合。

Rule1：减少继承、多用组合。

- 工厂模式、Facade模式、Builder模式……

Rule2：减少类型的成员多：

- 发消息通知
- 管道（组合）
- continuations

开闭原则

对扩展开放，对修改封闭。

提升程序的扩展性（比如插件、元数据、DSL等），减少对程序的修改。

领域设计原则

Rule1： 创建属于自己的领域方言，让每个对象拥有贴近场景的具体含义，做到专对象专用。

例如：HTML、JSX、@skedo/meta/Node，@skedo/creator/Project,

Rule2： 用DSL描述你的系统

DSL(json ,yml, builder) -> ActivityPage

Rule3：为不同的目标设计Context

参见:Skedo的RenderContext

Rule4 : 让元数据可以被扩展能力（插件、组件等）使用

参见@skedo的外部组件设计