React 实践思想

郭庆@tencent

腾讯云

1 现状

后之视今，亦犹今之视昔

腾讯云

1.1 使用 React 的一些作品

应用

- 云数据库产品（MySQL、TDSQL、Redis、MongoDB、..DB）
- 大数据可视化产品（云图）
- 以及智慧零售、私有化、各种工具……等产品

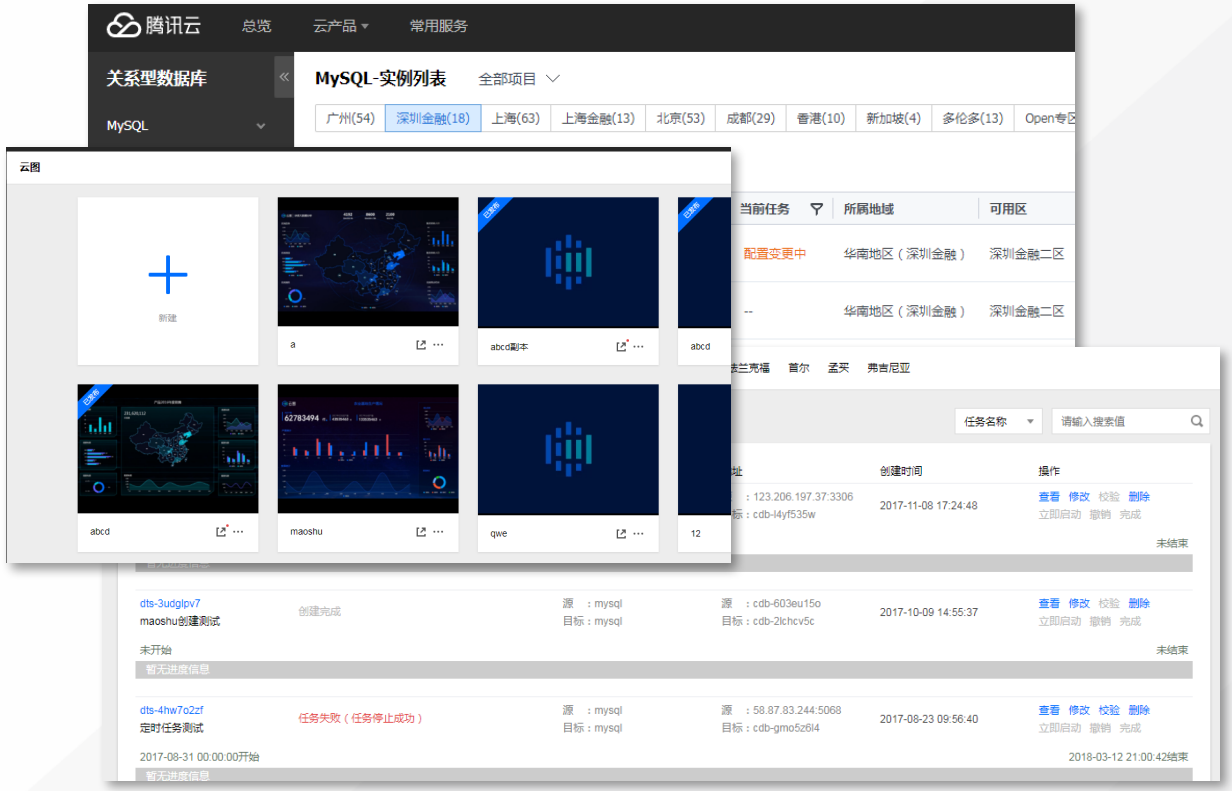
库/工具

- Bere
- QBT

框架

- TCFF

以及 脚手架、依赖管理工具、内部系统 等



1.2 React 技术栈

- React/Redux
- TCFF、Bere
- Redux-Saga，Lodash/fp、RxJS 还有其他

2 思想与实践

Don't make me think

2.2 实践一

一个（业务团队）组件库：Bere

2.2 一个（业务团队）组件库

Basic UI 组件

- 遵循设计规范，覆盖大部分交互场景
- 例如 按钮、日期时间选择器、数据表格、单选组 等

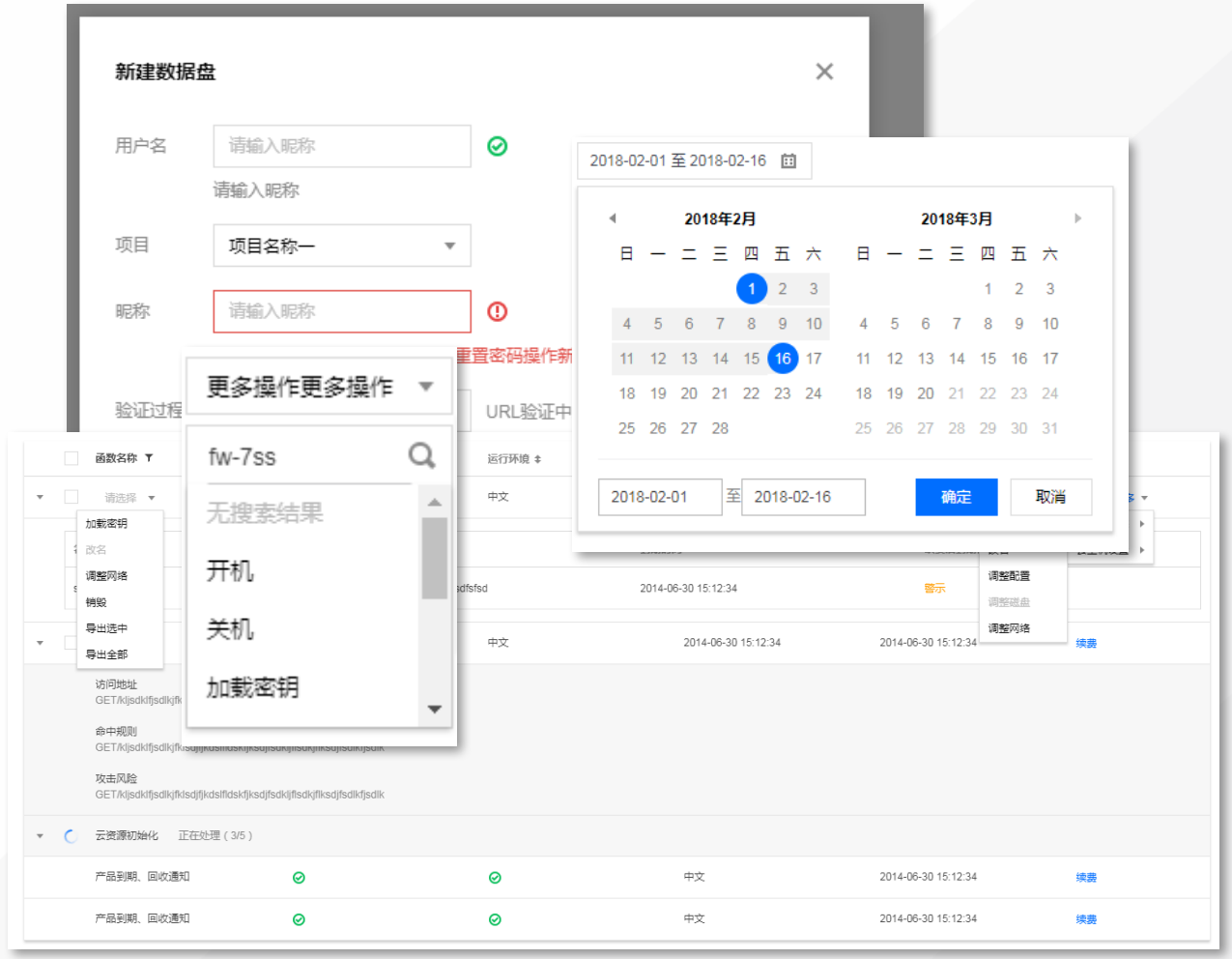
Functional 组件

- 侧重于提供功能而非外观
- 例如路由、滚动检测、表单验证 等

Behavior 组件

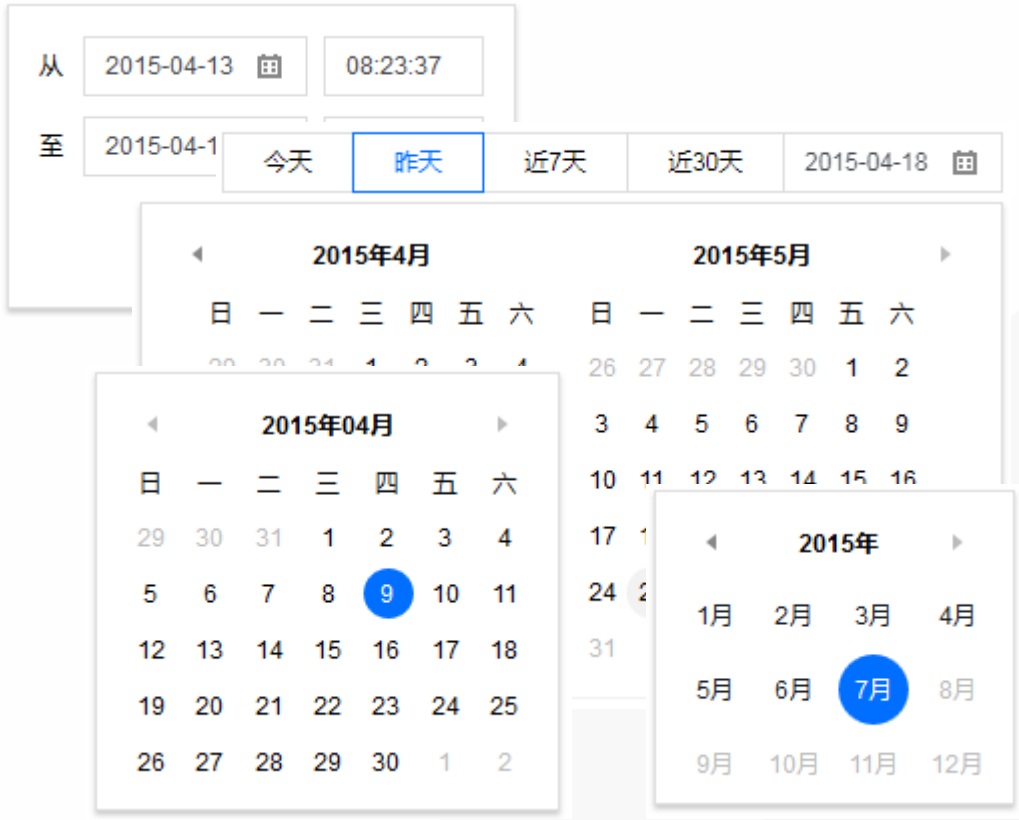
- 一般以高阶组件方式提供
- 提供某种能力，例如 Polling、WindowResize、Clickoutside 等

以及 各项目业务组件生态库



2.2 组件设计-问题1

一个时间组件的故事



2.2 组件设计-思想1

单一职责（The Single Responsibility Principle）

2.2 组件设计-问题2

一个路由组件的故事

1

```
<Route
  path={match.url + '/carnitas'}
  component={Carnitas}
```

2

```
<AppEntry
  path="/report/:type?"
  layout={SecondaryLayout}
  title="Report"
  component={cb => window.require.async('system.report', cb)}
```

```
<AppEntry
  path="/detail"
  layout={ManageLayout}
  component={() => import(/* webpackChunkName: "detail" */'./modules/detail')} />
```

2.2 组件设计-思想2

外观模式（Facade Pattern）

2.2 组件设计-问题3

一个表单组件的故事

```
<RadioGroup
  value={this.state.radioGroupValue}
  onChange={(value) => {
    this.setState({
      radioGroupValue: value
    })
  }}
  list={[{text: 'Text 1', value: 0}, {text: 'Text 2', value: 1}]}
/>
<CheckboxGroup
  value={this.state.checkboxValue}
  onChange={(c) => {
    this.setState({
      checkboxValue: c
    })
  }}
  limit={2}
  list={[{text: 'test1', value: 1}, {text: 'test2', value: 2}, {text:
/>
<DateTimeRangePickerEx
  tabs={tabs}
  value={this.state.dateTime}
  range={{
    maxLength: '30d'
  }}
  onChange={(value) => this.setState({
    dateTime: value
  })}
/>
```

```
  <StaticFormGroup
    label="账号名">
    {values.user}
  </StaticFormGroup>
  <FormGroupField
    name="password"
    label="密码"
    type="password"
    required
    component={Input}
    validate={[required, normalPassword]}
    placeholder="请输入帐号密码"
    text="密码需要 8 - 16 个字符"
  />
  <FormGroupField
    name="type"
    label="类型"
    required
    component={RadioGroup}
  />
  <FormGroupField
    name="rememberMe"
    label="记住我"
    component={Switch}
  />
</FormGroupList>
```

2.2 组件设计-思想3

里氏替换原则（The Liskov Substitution Principle）

2.3 实践二

一个（业务团队）前端应用框架：TCFF

2.3 一个（业务团队）的框架

主要能力

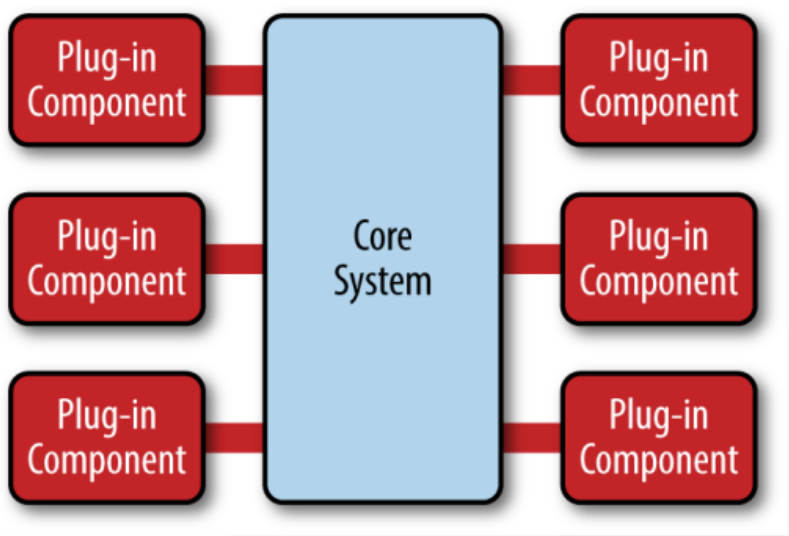
- 提供应用开发的整体能力（All in one）
- 让应用正确分层（Layered）
- 让应用正确模块化（Modular）

核心模式

- 单向数据流（Unidirectional data flow）
- 钩子 & 面向切面（Hooking、AOP）
- 依赖注入（DI）
- 插件化（Pluggable）

设计思想

- 少既是多（Less is more）



微内核架构

2.3 有了框架后所面临的问题

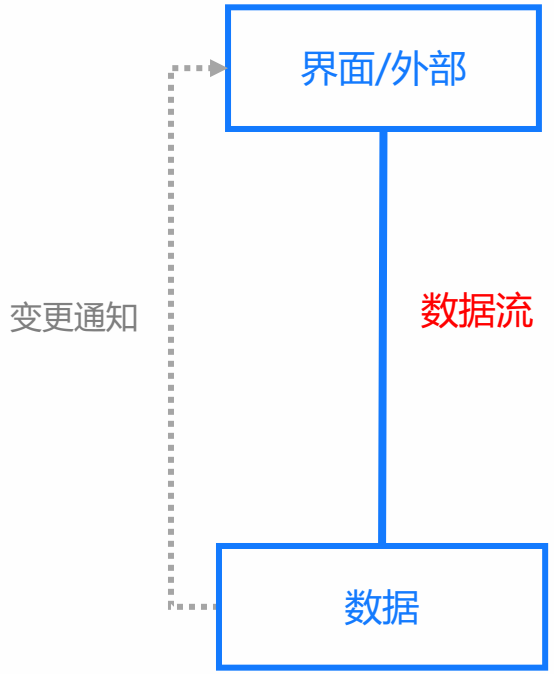
- 如何降低开发人员接入成本？
- 如何增强扩展性，满足不同需求？
- 如何显著提高开发人员效率和质量？

2.3 只举一个例子

单向数据流的进化史

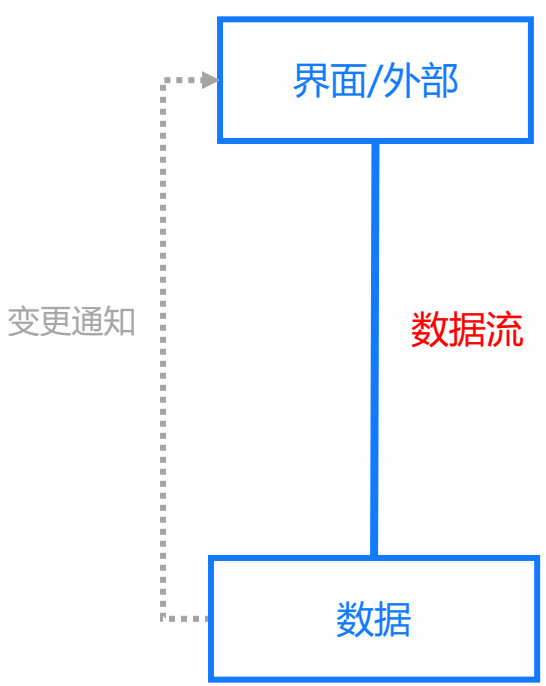
2.3 单向数据流模型

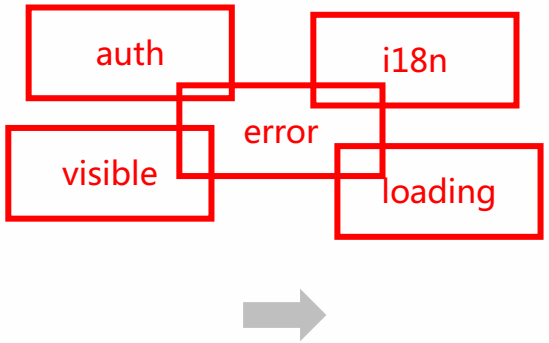
如何缩小问题规模？

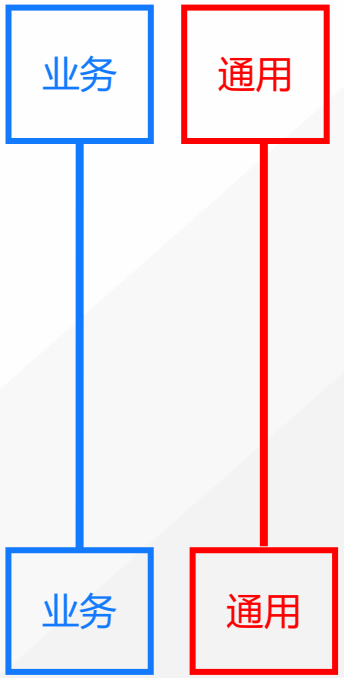


关注点分离（Separation of concerns，SOC）

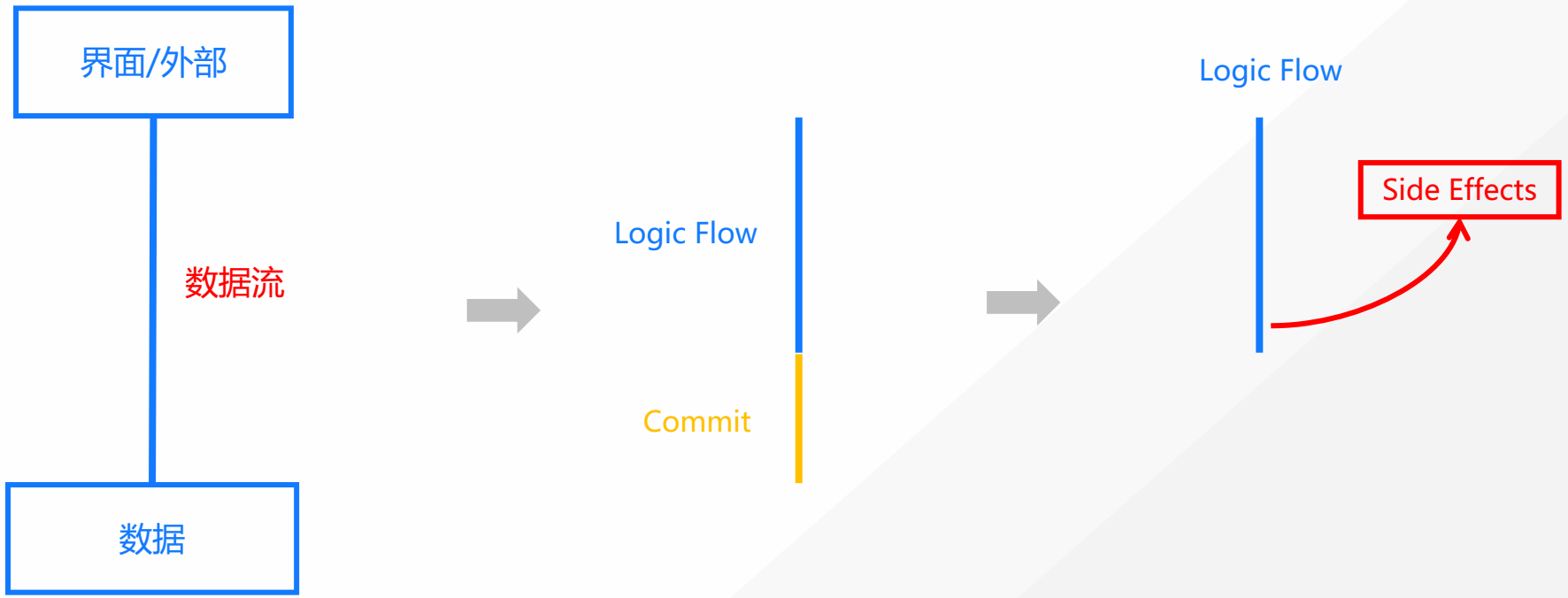
2.3 垂直分离



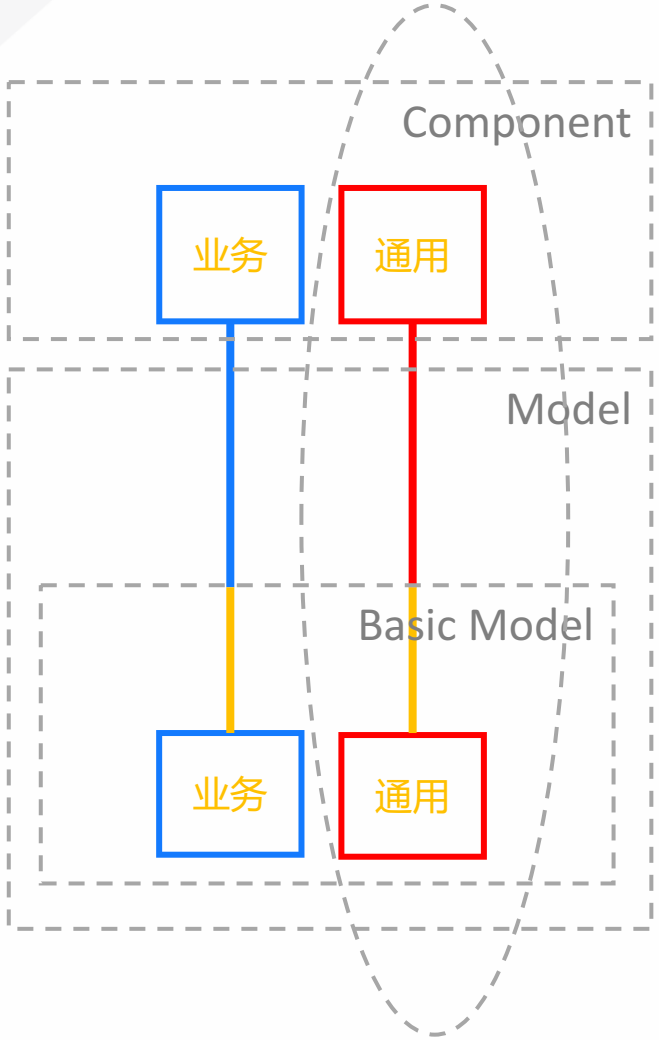


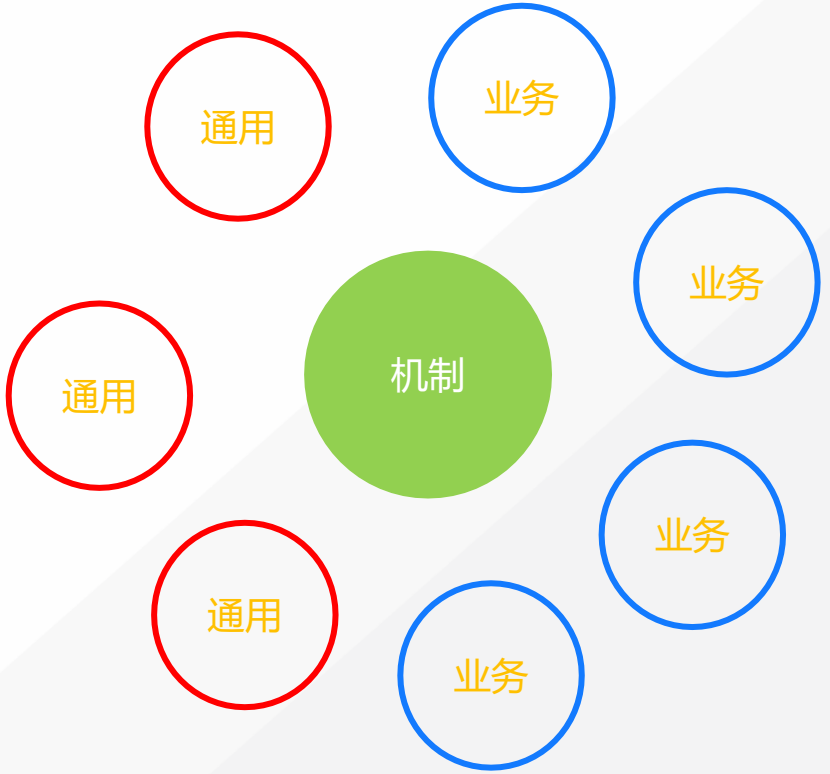


2.3 水平分离



2.3 实现 - 微内核的本质





2.3 Logic Flow 复杂性

Logic Flow

- 编程风格（imperative、functional、composition、fluent……）
- 异步逻辑（concurrence、sequence、race……）
- 业务逻辑（跨模型调用、数据共享……）
- 副作用处理：隔离、提纯

2.2 Logic Flow 指导思想

开闭原则（Open/closed principle）

2.2 Logic Flow 实现方案

逻辑流插件族（ Logic flow plugin family ）

Flow

签名

(action, inject) =>

接口 & 钩子

- getFlow
- runFlow
- onFlow

2.5 举个例子

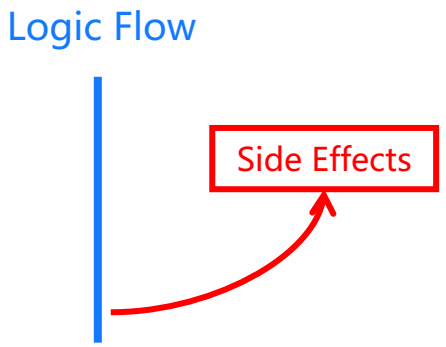
```
fetch: function * ({payload}, {select, call, put, callAction, context: {service}}) {
  const params = yield select(state => {
    return {
      groupId: state.group.currentId,
      perPage: state.user.pageSize,
      page: state.user.page
    }
  })

  const {resp} = yield call(service.getUserList, params)
  if (resp.success) {
    yield put({
      list: resp.data.users || [],
      total: resp.data.total || 0,
    })

    yield callAction({ type: 'fetchStat' })
  }
},
```

redux-saga

redux-observable

```
changeName: function (action$, {store, context: {service}}) {
  return action$
    .pipe(
      delay(1000),
      map(() => {
        const { name } = store.getState().user
        return {
          type: 'user/change',
          payload: {
            name
          }
        }
      })
    )
}
```

2.3 等等……还有副作用



谈谈“业务”框架的核心价值

2.3 副作用的抽象：Service

用 Service 为各种副作用、『脏东西』找到合理的归处

Service 的能力

- 开箱即用
- 加入 hook（onAjaxDone、onBeforeAjax……）
- 可扩展（RPC、RESTful、GraphQL……）
- 符合框架主流价值（error、message、防御内置）

```
getUserList: 'get /api/user/list',
modifyUserName: 'post /api/user/changeName',
deleteUser: 'post /api/user/delete',
```

```
service.getUserList
service.modifyUserName
service.deleteUser
```

3 总结

让你害怕的不是未知的未来，而是不断重复过去的错误

腾讯云

3.1 回顾

- 现状：产品和工具、React + Redux 全家桶
- 组件化设计：S.O.L.I.D 原则、设计模式
- 框架设计：少即是多、数据流开发优化

THANKS

腾讯云