

# JavaScript 设计模式及在业务中的应用

1. 十大设计原则
2. 设计模式在业务中的应用

AI 课增长售卖前端  
王鹏

# 十大设计原则

# 1. SRP 单一职责原则

- 定义：一个类/函数只负责完成一个职责或者功能
- 解读：
  - 类的职责变化往往是导致类变化的原因
  - 不仅是类，函数（方法）也要遵循单一职责原则
- 优点：如果类与方法的职责划分得很清晰，不但可以提高代码的可读性，更实际地降低了程序出错的风险，因为清晰的代码会让bug无处藏身，也有利于bug的追踪，也就是降低了程序的维护成本

# bad

```
● ● ●  
export default class Course {  
    public name: String = ""; // 课程名称  
    public intro: String = ""; // 课程介绍  
  
    classBegin() {  
        console.log(` ${this.name} 开始上课了`);  
    }  
  
    saleCourse() {  
        console.log(` ${this.name} 正在售卖`);  
    }  
}
```

# good

```
// Course.ts  
export default class Course {  
    public name: String = ""; // 课程名称  
    public intro: String = ""; // 课程介绍  
}  
  
// ICourse  
interface ICourse {  
    name: String;  
    intro: String;  
}  
  
// Teacher.ts  
export default class Teacher {  
    classBegin(course: ICourse) {  
        console.log(` ${course.name} 开始上课了`);  
    }  
}  
  
// Salesman.ts  
export default class Salesman {  
    saleCourse(course: ICourse) {  
        console.log(` ${course.name} 正在售卖`);  
    }  
}
```

## 2. OCP 开闭原则

- 定义：一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。
- 解读：用抽象构建框架，用实现扩展细节。不以改动原有代码的方式来实现新需求，而是应该以实现事先抽象出来的接口（或具体类继承抽象类）的方式来实现。
- 优点：实践开闭原则的优点在于可以在不改动原有代码的前提下给程序扩展功能。增加了程序的可扩展性，同时也降低了程序的维护成本。

# bad

```
● ● ●  
export default class Course {  
    public name: String = ""; // 课程名称  
    public intro: String = ""; // 课程介绍  
    public teacher: String = ""; // 讲师姓名  
    public text: String = ""; // 文字内容  
    public videoUrl: String = ""; // 视频地址  
    public audioUrl: String = ""; // 音频地址  
}
```

# good

```
// 课程基类  
export default class Course {  
    public name: String = ""; // 课程名称  
    public intro: String = ""; // 课程介绍  
    public teacher: String = ""; // 讲师姓名  
}  
  
// 文字课程  
import Course from "./Course";  
  
export default class TextCourse extends Course {  
    public text: String = "";  
  
    constructor() {  
        super();  
    }  
}  
  
// 视频课程  
import Course from "./Course";  
  
export default class VideoCourse extends Course {  
    public videoUrl: String = "";  
  
    constructor() {  
        super();  
    }  
}  
// .....
```

### 3. LSP 里式替换原则

- 定义：所有引用基类的地方必须能透明地使用其子类的对象，也就是说子类对象可以替换其父类对象，而程序执行效果不变。
- 解读：在继承体系中，子类中可以增加自己特有的方法，也可以实现父类的抽象方法，但是不能重写父类的非抽象方法，否则该继承关系就不是一个正确的继承关系。
- 优点：可以检验继承使用的正确性，约束继承在使用上的泛滥。



```
// IRect.ts
export default interface IRect {
    width: Number;
    height: Number;

    setWidth(w: Number): void;
    setHeight(h: Number): void;

    getArea(): Number;
}

// RectImpl.ts
import IRect from "./IRect";

export default class RectImpl implements IRect {
    width: number;
    height: number;

    setWidth(w: number): void {
        this.width = w;
    }
    setHeight(h: number): void {
        this.height = h;
    }
    getArea(): number {
        return this.width * this.height;
    }
}
```

bad • •

```
// ISquare.ts
import IRect from "./IRect";

export default interface ISquare extends IRect {}

// SquareImpl.ts
import ISquare from "./ISquare";

export default class SquareImpl implements ISquare {
    width: number;
    height: number;

    setWidth(w: number): void {
        this.width = w;
        this.height = w;
    }
    setHeight(h: number): void {
        this.height = h;
        this.width = h;
    }
    getArea(): number {
        return this.width * this.height;
    }
}
```



```
// 调用类
import RectImpl from "./lsp/bad/RectImpl";
import SquareImpl from "./lsp/bad/SquareImpl";

const rect = new RectImpl();
rect.setWidth(10);
rect.setHeight(20);

const square = new SquareImpl();
square.setWidth(10);
square.setHeight(20);

// 200, 400
console.log(rect.getArea(), square.getArea());
```

# good



```
// IQuadrangle.ts
export default interface IQuadrangle {
    width: number;
    height: number;

    setWidth(w: number): void;
    setHeight(h: number): void;

    getArea(): number;
}
```

```
// IRect.ts
import IQuadrangle from "./IQuadrangle";

export default interface IRect extends IQuadrangle {}

// ISquare.ts
import IQuadrangle from "./IQuadrangle";

export default interface ISquare extends IQuadrangle {}
```



```
// RectImpl.ts
import IRect from "./IRect";

export default class RectImpl implements IRect {
    width: number;
    height: number;
    setWidth(w: number): void {
        this.width = w;
    }
    setHeight(h: number): void {
        this.height = h;
    }
    getArea(): number {
        return this.width * this.height;
    }
}

// SquareImpl.ts
import ISquare from "./ISquare";

export default class SquareImpl implements ISquare {
    width: number;
    height: number;
    sideLen: number;

    setWidth(w: number): void {
        this.sideLen = w;
    }
    setHeight(h: number): void {
        this.sideLen = h;
    }
    getArea(): number {
        return this.sideLen * this.sideLen;
    }
}
```

## 4. ISP 接口隔离原则

- 定义：调用方不应该被强迫依赖它不需要的接口。
- 解读：“接口”的4种不同理解：
  - 一组接口集合
  - 单个api接口或函数
  - OOP中的接口
  - 组件对外暴露的props
- 优点：避免同一个接口里面包含不同类职责的方法，接口责任划分更加明确，符合高内聚松耦合的思想。

```
namespace Car {
    interface ICar {
        run(): void;
        drift(): void;
        cruise(): void;
        calcRoutes(): void;
    }

    class BasicCar implements ICar {
        run() {
            console.log("可以运行");
        }
        drift() {
            console.log("可以漂移");
        }
        cruise() {
            console.log("可以定速巡航");
        }
        calcRoutes() {
            console.log("可以自动计算路线");
        }
    }

    class PremiumCar implements ICar {
        run() {
            console.log("可以运行");
        }
        drift() {
            console.log("可以漂移");
        }
        cruise() {
            console.log("可以定速巡航");
        }
        calcRoutes() {
            console.log("可以自动计算路线");
        }
    }
}
```

# bad

```
class WulingHongguang {
    start(i: ICar): void {
        i.run();
    }
    turn(i: ICar): void {
        i.drift();
    }
}

class Tesla {
    start(i: ICar): void {
        i.run();
    }
    curise(i: ICar): void {
        i.cruise();
    }
    calcRoutes(i: ICar): void {
        i.calcRoutes();
    }
}

class Test {
    main() {
        const wuling = new WulingHongguang();
        wuling.start(new BasicCar());

        const tesla = new Tesla();
        tesla.calcRoutes(new PremiumCar());
    }
}
```

good

```
namespace Car {  
    interface ICar {  
        run(): void;  
    }  
  
    interface IBasicCar {  
        drift(): void;  
    }  
  
    interface IPremiumCar {  
        cruise(): void;  
        calcRoutes(): void;  
    }  
  
    class BasicCar implements ICar, IBasicCar {  
        run() {  
            console.log("可以运行");  
        }  
        drift() {  
            console.log("可以漂移");  
        }  
    }  
  
    class PremiumCar implements ICar, IPremiumCar {  
        run() {  
            console.log("可以运行");  
        }  
        cruise() {  
            console.log("可以定速巡航");  
        }  
        calcRoutes() {  
            console.log("可以自动计算路线");  
        }  
    }  
}
```

```
class WulingHongguang {  
    start(i: ICar): void {  
        i.run();  
    }  
    turn(i: IBasicCar): void {  
        i.drift();  
    }  
}  
  
class Tesla {  
    start(i: ICar): void {  
        i.run();  
    }  
    curise(i: IPremiumCar): void {  
        i.cruise();  
    }  
    calcRoutes(i: IPremiumCar): void {  
        i.calcRoutes();  
    }  
}  
  
class Test {  
    main() {  
        const wuling = new WulingHongguang();  
        wuling.start(new BasicCar());  
  
        const tesla = new Tesla();  
        tesla.calcRoutes(new PremiumCar());  
    }  
}
```

# 5. DIP 依赖倒置原则

- 定义：
  - 依赖抽象，而不是依赖实现。
  - 抽象不应该依赖细节；细节应该依赖抽象。
  - 高层模块不能依赖低层模块，二者都应该依赖抽象。
- 解读：
  - 针对接口编程，而不是针对实现编程。
  - 尽量不要从具体的类派生，而是以继承抽象类或实现接口来实现。
  - 关于高层模块与低层模块的划分可以按照决策能力的高低进行划分。业务层自然就处于上层模块，逻辑层和数据层自然就归类为底层。
- 优点：通过抽象来搭建框架，建立类和类的关联，以减少类间的耦合性。而且以抽象搭建的系统要比以具体实现搭建的系统更加稳定，扩展性更高，同时也便于维护。



# bad

```
namespace Bad {  
    class BackEndDeveloper {  
        writeJava( ) {  
            console.log("我会写 Java");  
        }  
    }  
  
    class FrontEndDeveloper {  
        writeJavascript( ) {  
            console.log("我会写 JavaScript");  
        }  
    }  
  
    class Project {  
        public implement( ) {  
            const bed = new BackEndDeveloper();  
            const fed = new FrontEndDeveloper();  
  
            bed.writeJava();  
            fed.writeJavascript();  
        }  
    }  
}
```



# good

```
namespace Good {  
    interface IDeveloper {  
        develop(): void;  
    }  
  
    class BackEndDeveloper implements IDeveloper {  
        public develop() {  
            this.writeJava();  
        }  
        private writeJava( ) {  
            console.log("我会写 Java");  
        }  
    }  
  
    class FrontEndDeveloper implements IDeveloper {  
        public develop() {  
            this.writeJavascript();  
        }  
        private writeJavascript( ) {  
            console.log("我会写 JavaScript");  
        }  
    }  
  
    class Project {  
        private developers: Array<IDeveloper>;  
        constructor(developers: Array<IDeveloper>) {  
            this.developers = developers;  
        }  
        public implement( ) {  
            this.developers.forEach((d) => d.develop());  
        }  
    }  
}
```

# 6. LOD 最少知识原则（迪米特法则）

- 定义：一个对象应该尽可能少的与其他对象有接触，也就是只接触那些真正需要接触的对象。
- 解读：迪米特法则也叫做最少知识原则（Least Know Principle），一个类应该只和它的成员变量，方法的输入，返回参数中的类作交流，而不应该引入其他的类（间接交流）
- 优点：实践迪米特法则可以良好地降低类与类之间的耦合，减少类与类之间的关联程度，让类与类之间的协作更加直接。

## 7. KISS 原则

- keep it simple and stupid
- 概念：尽量保持简单
- 意义：保持代码可读和可维护的重要手段
- 如何实现：
  - 不要使用同时可能不懂的技术来实现代码
  - 不要重复造轮子
  - 不要过度优化

## 8. YAGNI 原则

- you aren't gonna need it
- 概念：不要去设计当前用不到的功能，不要去编写当前用不到的代码
- 核心：不要过度设计

## 9. DRY 原则

- Don't Repeat Yourself.
- 概念：不要写重复的代码、不要重复造轮子
- 代码重复的情况：
  - 实现逻辑重复
  - 功能语义重复
  - 代码执行重复

# 9. 组合/聚合复用原则 CRP

- 组合优于继承
- 尽可能通过组合已有对象的能力来实现新的功能，而不是使用继承来获取这些能力

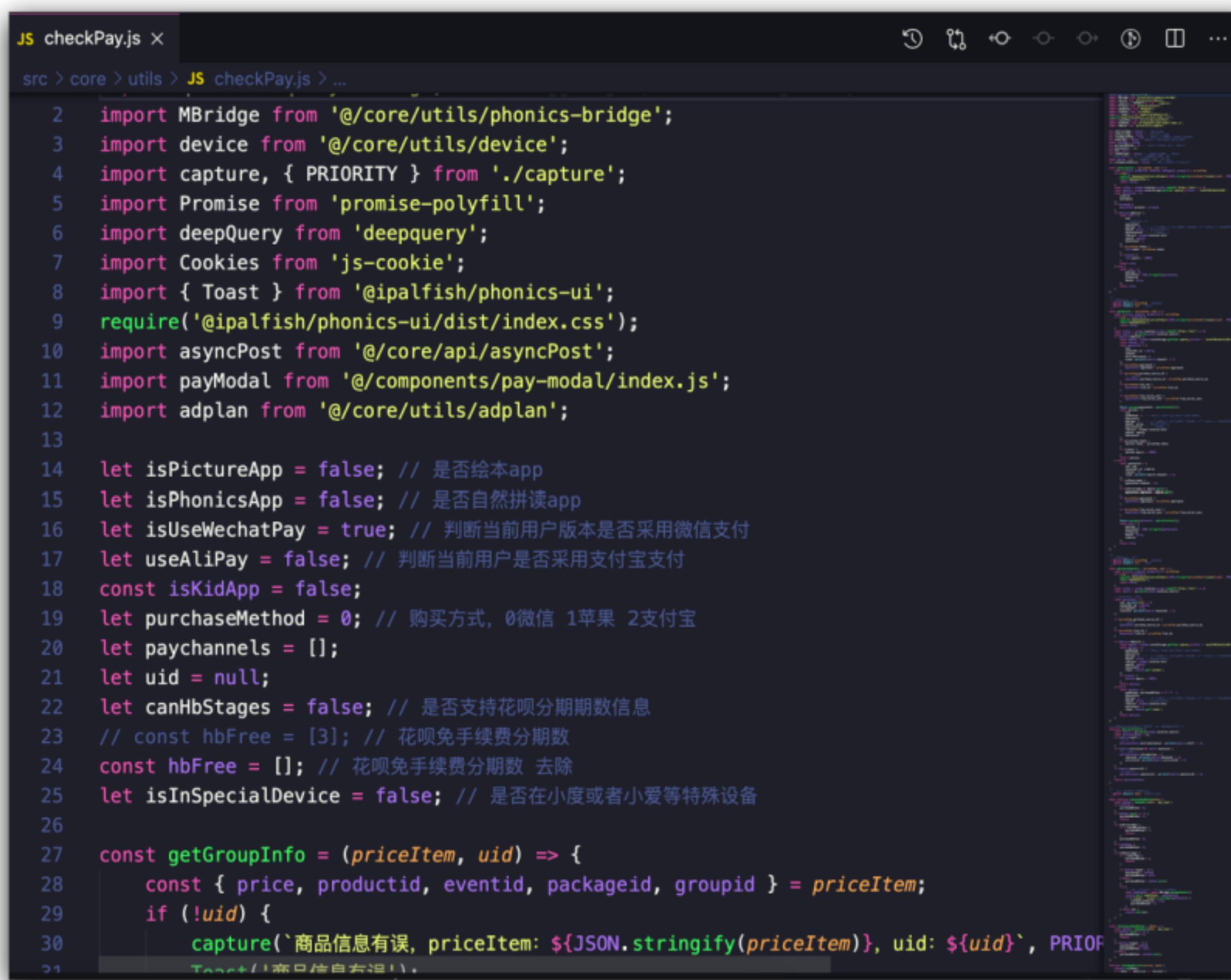
# 小结

- SOLID
  - S - Single Responsibility Principle 单一职责原则 (SRP)
  - O - Open Closed Principle 开闭原则 (OCP)
  - L - Liskov Substitution Principle 里氏替换原则 (LSP)
  - I - Interface Segregation Principle 接口隔离原则 (ISP)
  - D - Dependency Inversion Principle 依赖倒置原则 (DIP)
- 迪米特法则 (Law Of Demeter) 又叫做最少知识原则
- KISS 尽量保持简单
- YAGNI 不要过度设计
- DRY 不要写重复的代码
- CRP 组合优于继承

# 设计模式与业务相结合

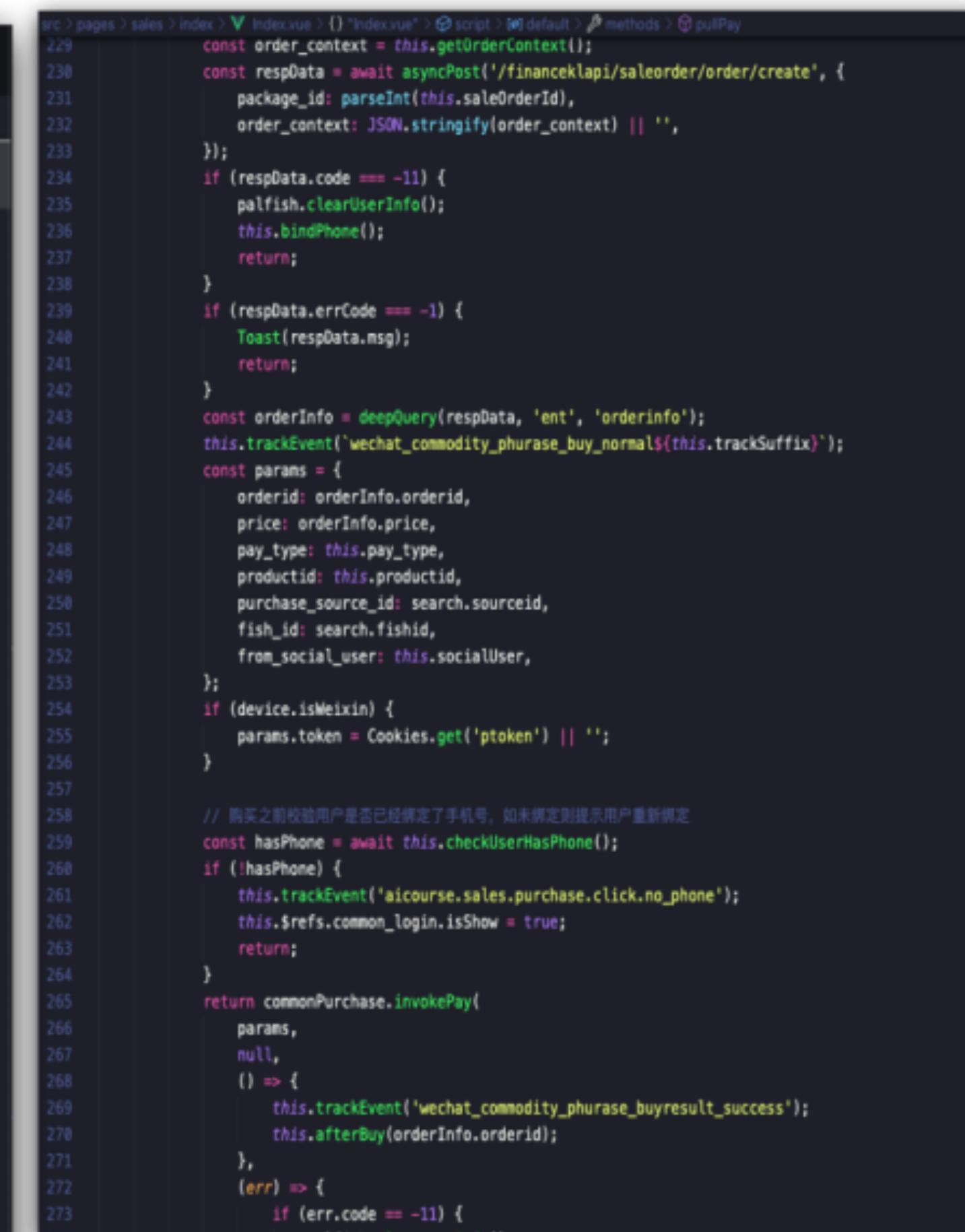
# 1. 基于工厂模式对支付模块进行重构

800行左右支付逻辑代码



```
JS checkPay.js X
src > core > utils > JS checkPay.js > ...
1 import MBridge from '@core/utils/phonic-bridge';
2 import device from '@core/utils/device';
3 import capture, { PRIORITY } from './capture';
4 import Promise from 'promise-polyfill';
5 import deepQuery from 'deepquery';
6 import Cookies from 'js-cookie';
7 import { Toast } from '@ipalfish/phonic-ui';
8 require('@ipalfish/phonic-ui/dist/index.css');
9 import asyncPost from '@core/api/asyncPost';
10 import payModal from '@components/pay-modal/index.js';
11 import adplan from '@core/utils/adplan';
12
13 let isPictureApp = false; // 是否绘本app
14 let isPhonicsApp = false; // 是否自然拼读app
15 let isUseWechatPay = true; // 判断当前用户版本是否采用微信支付
16 let useAliPay = false; // 判断当前用户是否采用支付宝支付
17 const isKidApp = false;
18 let purchaseMethod = 0; // 购买方式, 0微信 1苹果 2支付宝
19 let paychannels = [];
20 let uid = null;
21 let canHbStages = false; // 是否支持花呗分期期数信息
22 // const hbFree = [3]; // 花呗免手续费分期数
23 const hbFree = []; // 花呗免手续费分期数 去除
24 let isInSpecialDevice = false; // 是否在小度或者小爱等特殊设备
25
26 const getGroupInfo = (priceItem, uid) => {
27   const { price, productid, eventid, packageid, groupid } = priceItem;
28   if (!uid) {
29     capture(`商品信息有误, priceItem: ${JSON.stringify(priceItem)}, uid: ${uid}`, PRIORITY);
30     Toast('商品信息有误');
31   }
32   return { price, productid, eventid, packageid, groupid };
33 }
34
35 const handlePay = (method, params) => {
36   if (method === 'alipay') {
37     alipay();
38   } else if (method === 'wechat') {
39     wechat();
40   }
41 }
42
43 const alipay = () => {
44   // Alipay logic
45 }
46
47 const wechat = () => {
48   // WeChat Pay logic
49 }
50
51 const handleSuccess = (result) => {
52   // Success handling logic
53 }
54
55 const handleFailure = (err) => {
56   // Failure handling logic
57 }
58
59 const handleCancel = () => {
60   // Cancel handling logic
61 }
62
63 const handlePayResult = (result) => {
64   // Pay result handling logic
65 }
66
67 const handleRefund = (result) => {
68   // Refund handling logic
69 }
70
71 const handleRefundResult = (result) => {
72   // Refund result handling logic
73 }
74
75 const handleRefundSuccess = (result) => {
76   // Refund success handling logic
77 }
78
79 const handleRefundFailure = (err) => {
80   // Refund failure handling logic
81 }
82
83 const handleRefundCancel = () => {
84   // Refund cancel handling logic
85 }
86
87 const handleRefundResult = (result) => {
88   // Refund result handling logic
89 }
90
91 const handleRefundSuccess = (result) => {
92   // Refund success handling logic
93 }
94
95 const handleRefundFailure = (err) => {
96   // Refund failure handling logic
97 }
98
99 const handleRefundCancel = () => {
100   // Refund cancel handling logic
101 }
```

业务中调用时



```
src > pages > sales > Index > Index.vue > script > default > methods > pullPay
229 const order_context = this.getOrderContext();
230 const respData = await asyncPost('/financeklapi/saleorder/order/create', {
231   package_id: parseInt(this.saleOrderId),
232   order_context: JSON.stringify(order_context) || '',
233 });
234 if (respData.code === -11) {
235   palfish.clearUserInfo();
236   this.bindPhone();
237   return;
238 }
239 if (respData.errCode === -1) {
240   Toast(respData.msg);
241   return;
242 }
243 const orderInfo = deepQuery(respData, 'ent', 'orderinfo');
244 this.trackEvent('wechat_commodity_phurase_buy_normals${this.trackSuffix}');
245 const params = {
246   orderid: orderInfo.orderid,
247   price: orderInfo.price,
248   pay_type: this.pay_type,
249   productid: this.productid,
250   purchase_source_id: search.sourceid,
251   fish_id: search.fishid,
252   from_social_user: this.socialUser,
253 };
254 if (device.isWeixin) {
255   params.token = Cookies.get('ptoken') || '';
256 }
257
258 // 购买之前校验用户是否已经绑定了手机号, 如未绑定则提示用户重新绑定
259 const hasPhone = await this.checkUserHasPhone();
260 if (!hasPhone) {
261   this.trackEvent('aicourse.sales.purchase.click.no_phone');
262   this.$refs.common_login.isShow = true;
263   return;
264 }
265 return commonPurchase.invokePay(
266   params,
267   null,
268   () => {
269     this.trackEvent('wechat_commodity_phurase_buyresult_success');
270     this.afterBuy(orderInfo.orderid);
271   },
272   (err) => {
273     if (err.code === -11) {
274       // Handle error
275     }
276   }
277 );
```

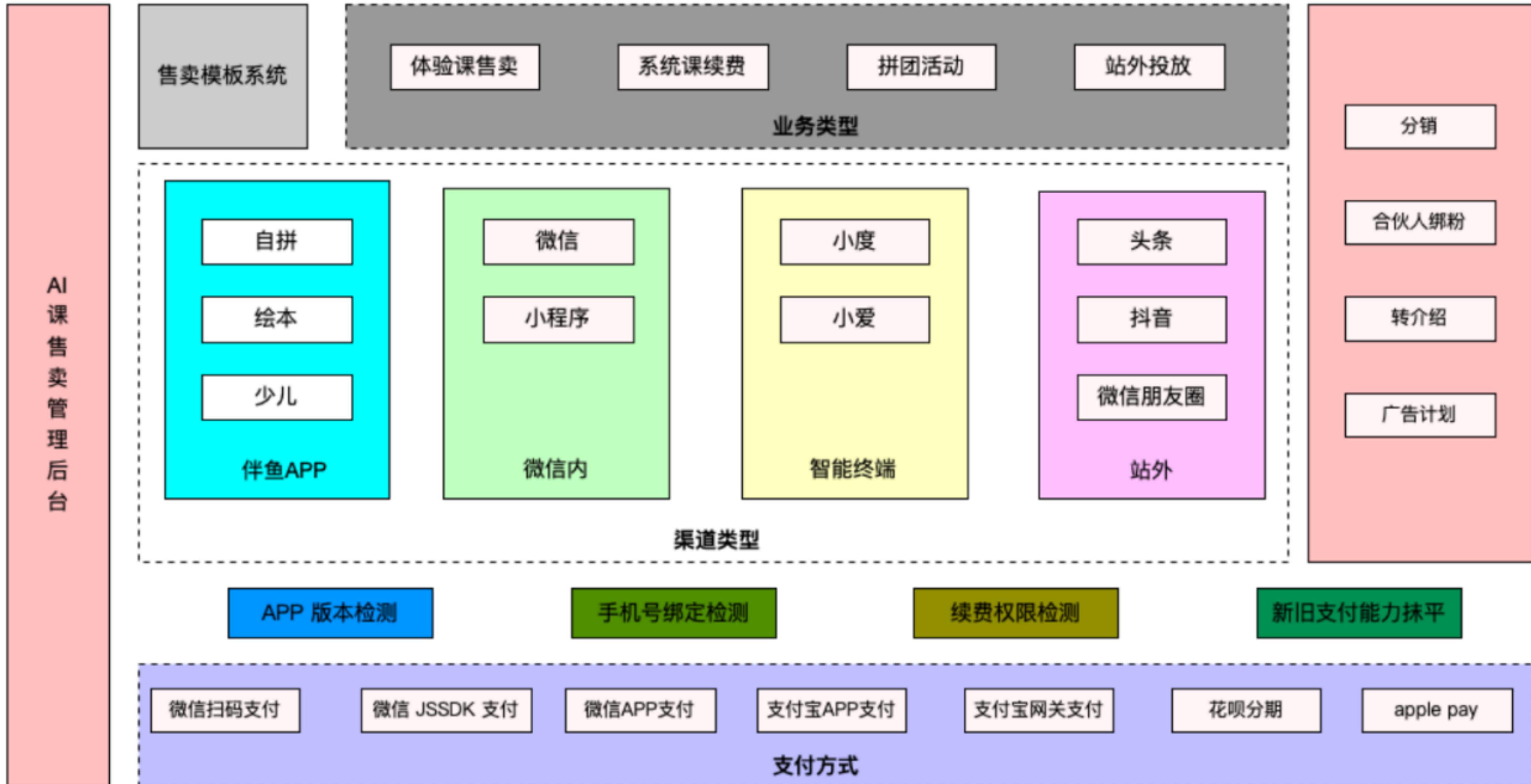
# 当前的支付模块有什么问题

- 各业务模块代码耦合严重
- 各方法之间重复逻辑较多
- 参数传递链路过长
- 业务页面中调用时不够简洁

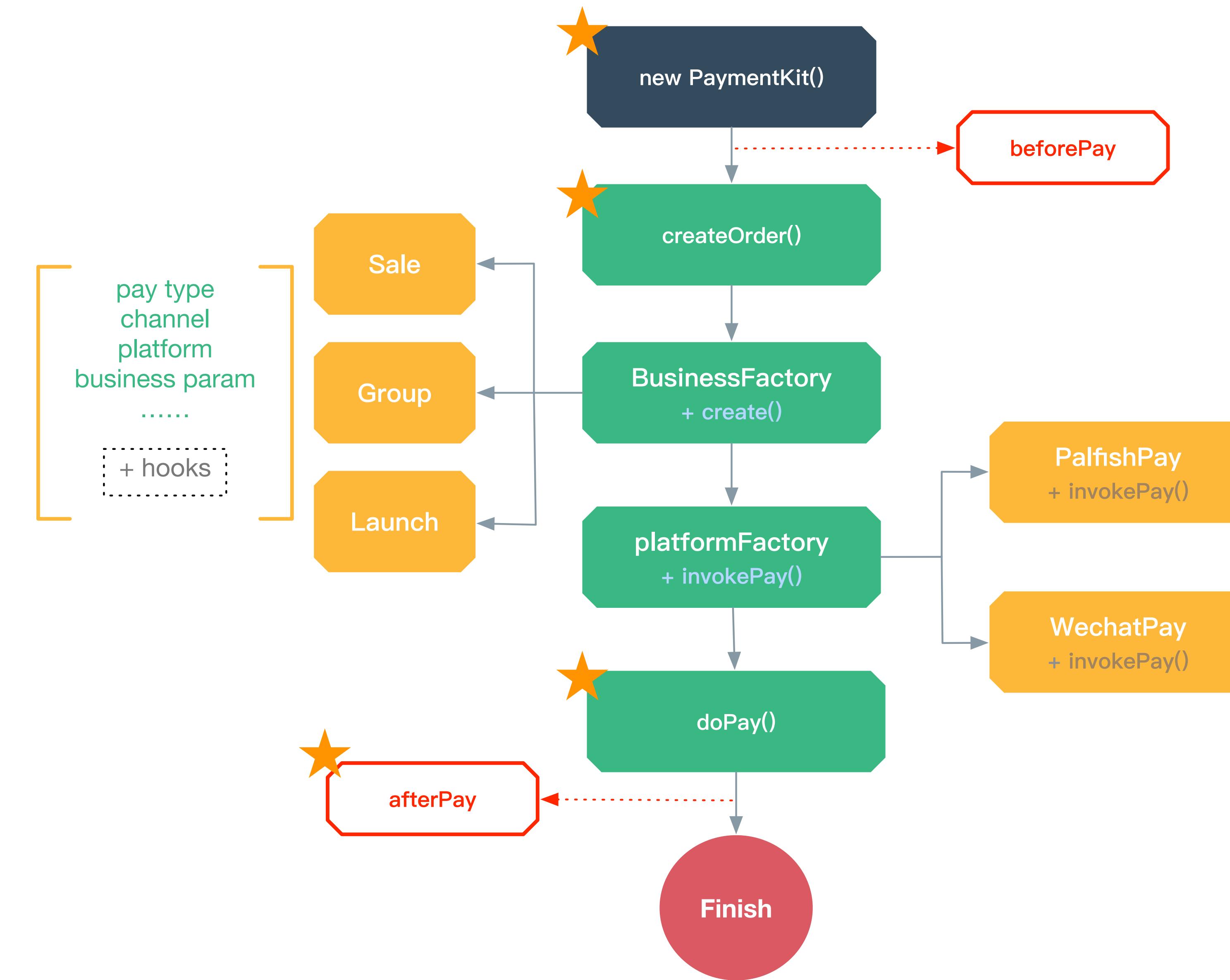
# 解决方案：工厂模式

- 定义：它提供了一种创建对象的最佳方式。使用工厂模式，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。
- 使用场景：
  - 调用方不知道如何选择接口（需要动态地决定使用哪个接口）
  - 几个类的主要逻辑相同，但是部分逻辑的算法和实现上稍有差别

## AI 课售卖 —— 支付模块架构图



# 支付流程调用逻辑图 ver 1.0



## 重构前业务页面中的调用 (168行)

```
/** 滚动支付 */
async pullPay() {
    if (!this.saleorderId) {
        captureSaleorderId('支付订单ID为空: ${this.saleorderId}', PRIORITY.PO);
        Toast('网络故障, 请刷新页面重试');
        return;
    }
    // 检测是否已购
    const { is_canyay, needLogin, reason } = await this.$checkPrebuy(this.commodityId);
    if (!is_canyay) {
        if (needLogin) {
            palish.clearUserInfol();
            this.bindPhone();
            return;
        }
        Toast(reason || '请勿重复购买');
        return;
    }
    // 检查是否已付
    const hasPhone = await this.$checkUserHasPhone();
    if (hasPhone) {
        this.$trackEvent('alcourse_sales.purchase.click_no_phone');
        this.$refs.common_login.$show = true;
        return;
    }
    // 生成订单
    const order_context = this.getOrderByContext();
    const respdata = await asyncPost(
        '/financelap/saleorder/order/create',
        package_id: parseInt(this.saleorderId),
        order_context: JSON.stringify(order_context) || '',
        true
    );
    if (respdata.code === -11) {
        palish.clearUserInfol();
        this.bindPhone();
        return;
    }
    if (respdata.errCode === -1) {
        Toast(respdata.msg);
        return;
    }
    const orderInfo = deepQuery(respdata, 'ent', 'orderInfo');
    this.$trackEvent('wechat_commodity_phurase_buy_normal${this.trackSuffix}');
    const params = {
        orderid: orderInfo.orderid,
        price: orderInfo.price,
        pay_type: orderInfo.pay_type,
        productid: this.productid,
        purchase_source_id: search.sourceid,
        fish_id: search.fishid,
        from_social_user: this.socialUser,
    };
    if (device.isWeixin) {
        params.token = Cookies.get('ptoken') || '';
    }
    return commonPurchase.invokePay(
        params,
        null,
        () => {
            this.$trackEvent('wechat_commodity_phurase_buyresult_success');
            this.$refs.bindPhone();
            return;
        },
        () => {
            if (err.code === -11) {
                palish.clearOrderInfo();
                this.$refs.bindPhone();
                return;
            }
            if (err.reason === 'get_brand_wcpay_request:cancel') {
                // 支付取消
                this.$trackEvent('wechat_commodity_phurase_buyresult_cancel');
            } else {
                capture(pullPay, {JSON.stringify(err)}, PRIORITY.PO);
                this.$trackEvent('wechat_commodity_phurase_buyresult_failure');
            }
            this.$refs.showUnpayMsg = true;
        }
    );
}
getOrderByContext() {
    let channel = 8;
    if (this.$refs.IWXApp) {
        channel = 15;
    }
    if (this.$refs.IWPictureApp) {
        channel = 15;
    }
    if (this.$refs.IPhonicsApp) {
        channel = 13;
    }
    if (device.isWeixin) {
        channel = 8;
    }
    const orderContext = {
        launch_id: launchid,
        sponsor_uid: search.sponsorid || '0',
        channel,
    };
    if (search.sourceid) {
        orderContext.purchase_source_id = search.sourceid;
    }
    if (search.fishid) {
        orderContext.fish_id = search.fishid;
    }
    if (this.socialUser) {
        orderContext.from_social_user = this.socialUser;
    }
    if (search.channel && Number.isNaN(parseInt(search.channel))) {
        orderContext.scene = parseInt(search.channel);
    }
    if (search.promoteId && Number.isNaN(parseInt(search.promoteId))) {
        orderContext.promote_id = parseInt(search.promoteId);
    }
    if (search.promoteId && Number.isNaN(parseInt(search.promoteId))) {
        orderContext.promote_id = parseInt(search.promoteId);
    }
    return orderContext;
}
bindPhone() {
    if (!search.sponsorid) return;
    const params = {
        friends: parseInt(search.sponsorid),
    };
    post('/wgc/picturebook/wallet/initify/friend/set', params).catch((err) => {
        console.log(JSON.stringify(err));
    });
}
// 购买成功后处理
afterBuyOrder() {
    this.$trackEvent('alcourse_sales.sale_page.click_after_buy');
    const that = this;
    function setTimer() {
        let timer;
        post('/financelap/saleorder/order/detail', { orderid })
            .then(function (res) {
                const status = deepQuery(res, 'ent', 'status');
                if (status === 20) {
                    clearInterval(timer); // 清除定时器
                    that.$trackEvent('course_sales.sale_page.click_complete_wait');
                    Indicator.close();
                    window.location.replace(`#/course/${that.courseid}/origin/wechat_course/main/complete-wait.html?${qs.stringify({
                        courseid: that.courseid,
                        appid: search.appid,
                        channel: search.channel,
                        subject: that.course_subject,
                        coursePeriodTypeId: that.coursePeriodTypeId,
                    })}`);
                } else {
                    time = setTimeout(() => {
                        setTimer();
                    }, 1000);
                }
            })
            .catch(function (error) {
                Indicator.close();
                console.log(error);
            });
    }
    setTimer();
}

```

## 重构后mixin文件中 (61行)

```
import PaymentKit from '@/core/payment-kit';

const NOOP = () => {};
export default {
    methods: {
        /**
         * 通用支付方法
         * @param {String} bizType 业务类型
         * @param {String} isRenew 是否续费场景
         * @param {Function} loginCallback 登录方法
         * @param {String} orderContext 订单上下文
         * @returns
         */
        async commonPay(bizType, isRenew, loginCallback = NOOP, orderContext = {}) {
            // 支付模块实例化
            const paymentKit = new PaymentKit(bizType, {
                from_social_user: this.socialUser,
                ...orderContext,
            });

            // 优惠券信息
            let coupons = [];
            if (this.couponList) {
                coupons = this.couponList.map(parseInt);
            }

            // 订单创建
            const createCode = await paymentKit.createOrder(this.saleorderId, coupons);
            if (createCode === -11) {
                loginCallback();
                return;
            } else if (createCode !== 1) return;

            this.$trackEvent('wechat_commodity_phurase_buy_normal');

            // 调用支付
            paymentKit.doPay({
                productid: this.productid,
                pay_type: this.pay_type,
                isRenew,
            });

            // 支付后的结果监听
            paymentKit.emitter.on('afterPay', ({ code, orderid }) => {
                if (code === 1) {
                    this.$trackEvent('wechat_commodity_phurase_buyresult_success');
                    paymentKit.handlers.afterPay(
                        {
                            orderid,
                            courseid: this.courseid,
                            subject: this.course_subject,
                            coursePeriodTypeId: this.coursePeriodTypeId,
                        },
                        true
                    );
                } else {
                    this.$refs.showUnpayMsg = true;
                    this.$trackEvent('wechat_commodity_phurase_buyresult_failure');
                }
            });
        },
    },
};

this.commonPay('sale', false, this.bindPhone);
```

## 重构后业务页面中的调用 (1行)

## 2. 门面模式（外观模式）

- 定义：提供一个统一的接口去访问多个子系统的多个不同的接口，它为子系统中的一组接口提供一个统一的高层接口。使得子系统更容易使用。
- 使用场景：
  - 调用方只需要复杂系统的某个子集，或者需要以特殊方式与系统交互时
  - 希望封装和隐藏原系统时
  - 当需要跟踪原系统的整体使用情况时

## 页面模板编辑

### 首屏配置

头图:

为保障展示质量, 请上传小于300k的jpg或者jpeg

头部视频:

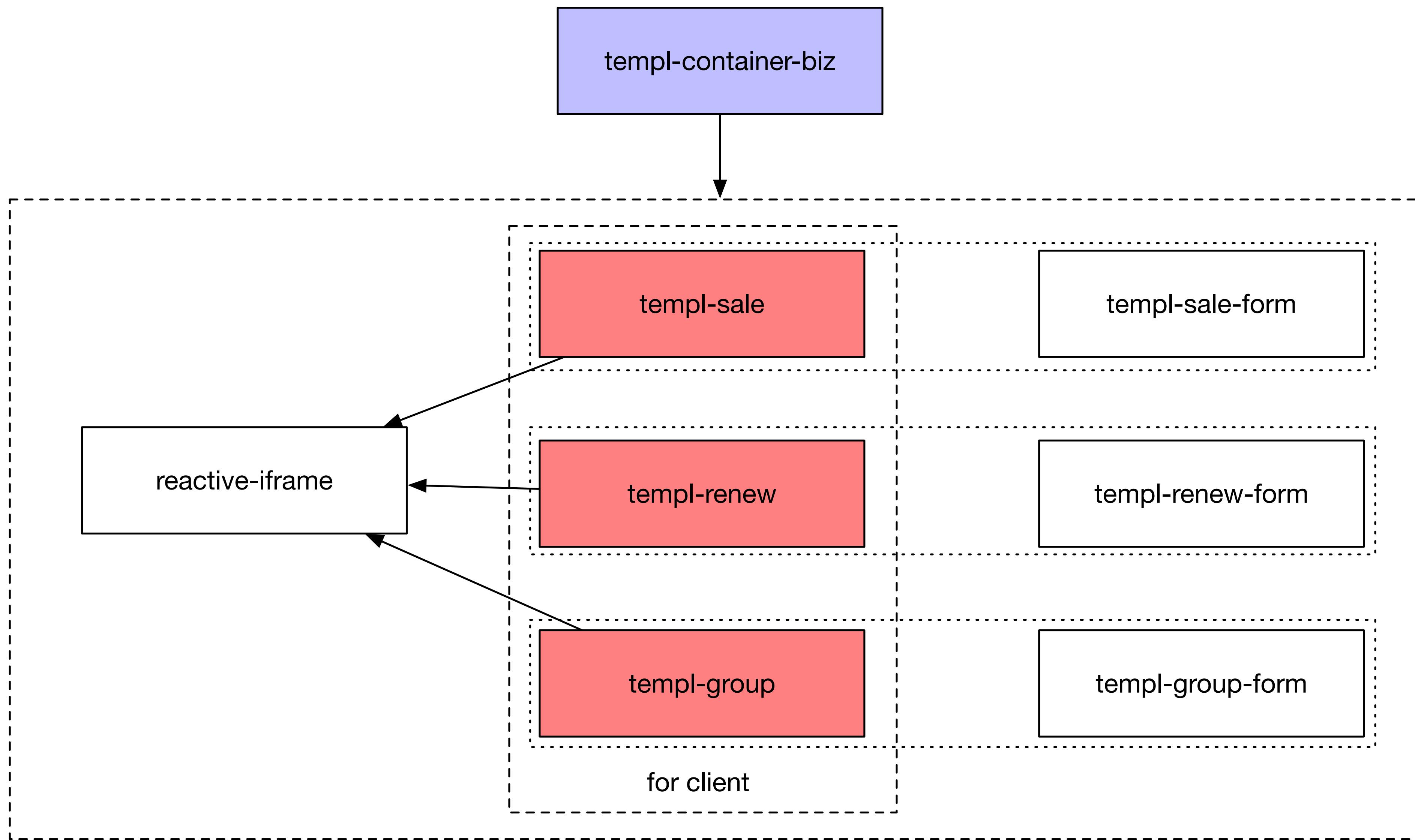
悬浮视频:

### 页面详情配置

图片配置:

为保障展示质量, 请上传小于300k的jpg或者jpeg

# 基于门面模式的售卖模板组件库



# Q&A

**Thank You!**