微保大型小程序的工程化实践

李锂

微保 高级架构师

自我介绍

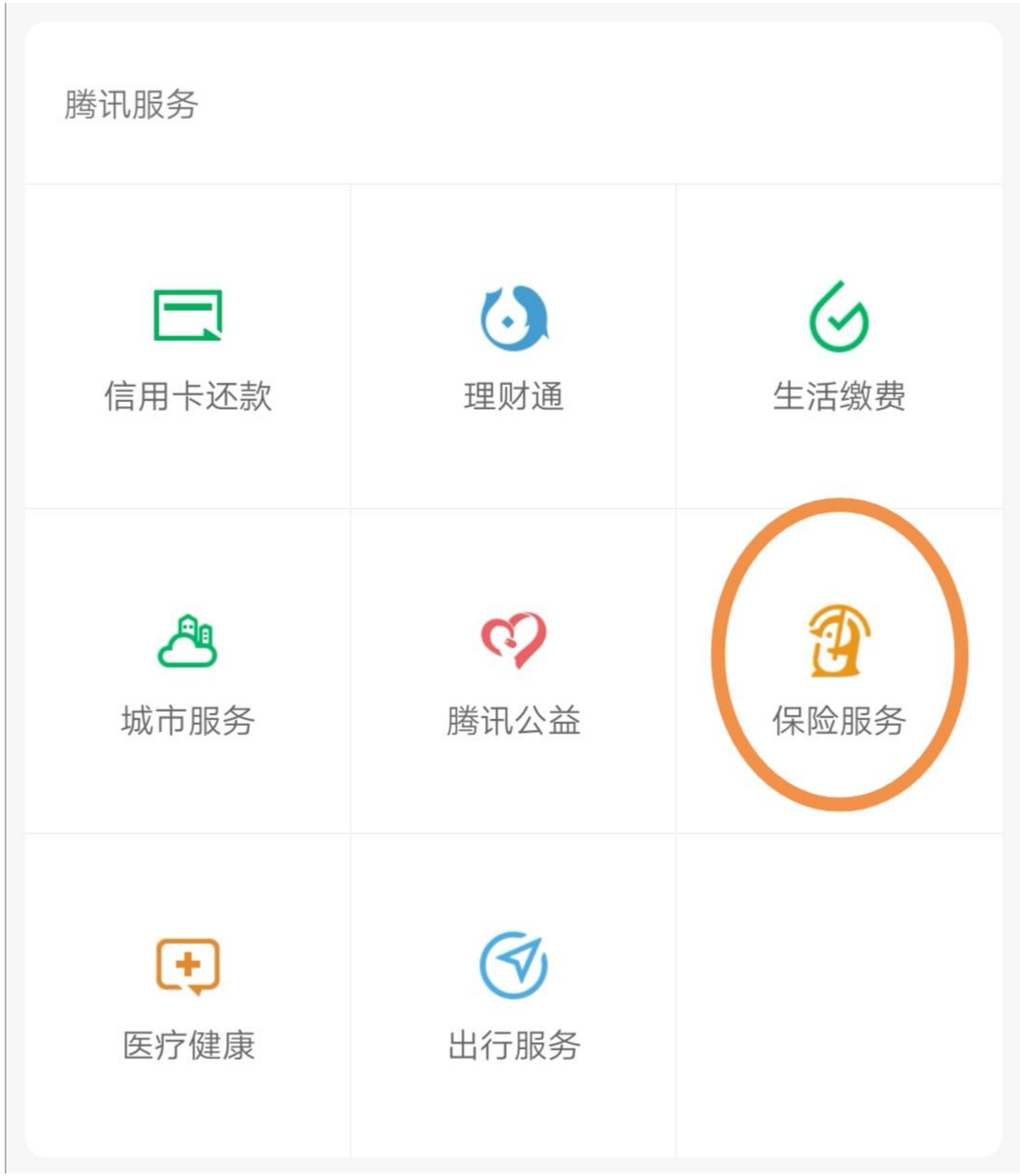
工作大部分时间在腾讯从事**搜索后台**相关的研发

2年的创业经历让自己成长为了一名**全栈**工程师

现就职于微保，主要负责**保险核心系统设计以及微保前端架构设计**

认识微保

- 腾讯旗下保险代理平台
- 微信支付九宫格“保险服务”入口
- 首个保险服务小程序，MAU超过3000万



目录

微保的早期版本

仅3款产品

1个小的前端团队

2周一个版本

包大小不到2M，名副其实的小程序

2017.12

微保的当前版本

4 个功能板块

5 个保险类目

40+ 个保险产品

15+ 个运营活动

4 个前端开发团队

大迭代2周，小迭代1周

包大小突破8M，巨无霸小程序

2019.10

大型小程序面临的挑战

1. 包大小限制
2. 包的编译速率
3. 协同开发效率
4. 数据加载性能
5. 发布的复杂性

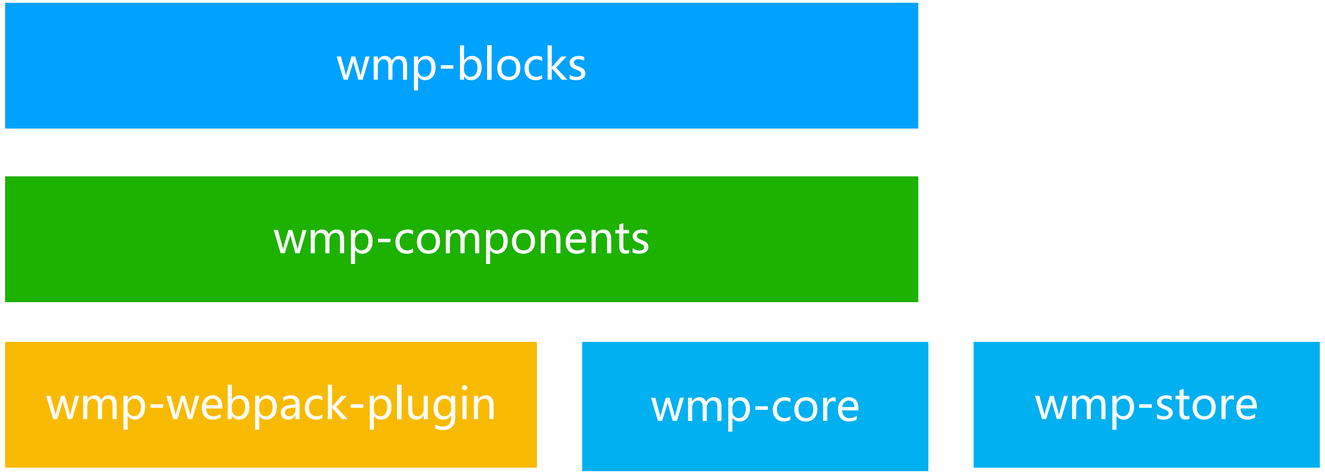
工程化的基础-小程序框架

为什么需要小程序框架

那么多开源框架，为什么我们还需要一个？

1. 微保小程序框架诞生较早
2. 微保小程序无需多端支持这个复杂特性
3. 理念上，微保小程序框架无意于提供类似 Vue 或 React 的开发方式

小程序框架设计



wmp-webpack-plugin: 工程化工具，单JS文件编译成多文件，支持模块化拆分

wmp-core: 收敛了公共技术特性，如：优化渲染性能，预加载能力，消息总线能力，埋点数据上报等

wmp-store: 使用 mobx 封装了数据状态管理

wmp-components: 视觉定义的基础组件封装

wmp-blocks: 业务组件封装，业务组件的引入可大大减少小程序包大小

工程化工具

```
import { page, api } from '@wmp/core';
import theme from '../theme';

page({
    config: {
        navigationBarBackgroundColor: theme.primaryColor,
    },

    style: {
        article: {
            backgroundColor: theme.primaryColor,
        },
    },

    template: (
        <view class="article">
            <wmp-markdown>{{text}}</wmp-markdown>
        </view>
    ),

    init() {
        this.contentAPI = api('content');
    },

    async onLoad({ contentID }) {
        const content = await this.contentAPI
          .cmd('getContent', { contentID })
          .call();
        this.text = content.text;
    }
});
```

```
// index.json
{
    "navigationBarBackgroundColor": "#ed7e2c",
    "usingComponents": {
        "wmp-markdown": "wmp-components/markdown"
    }
}
```

```
// index.wxss
.article {
    background-color: #ed7e2c
}
```

```
<!-- index.wxml -->
<view class="article">
    <wmp-markdown>{{text}}</wmp-markdown>
</view>
```

```
// index.js
import { page, api } from '@wmp/core';
page({
    onLoad({ contentID }) {
        const contentAPI = api('content');
        contentAPI.cmd('getContent', { contentID }).call().then(content => {
            this.text = content.text;
        });
    }
})
```

工程化工具

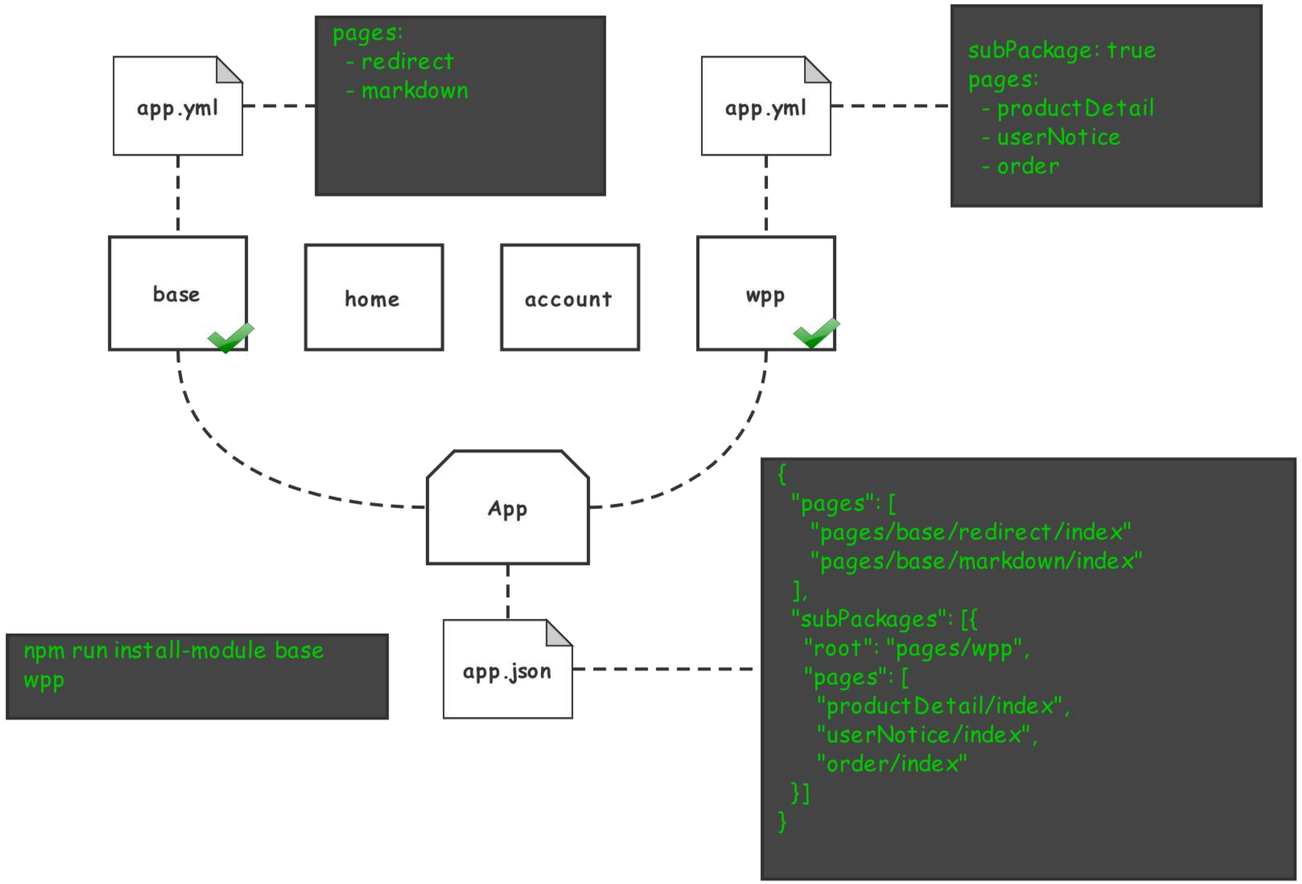
```
import { page, api } from '@wmp/core';
import theme from '../theme';

page({
    config: {
        navigationBarBackgroundColor: theme.primaryColor,
    },

    style: {
        article: {
            backgroundColor: theme.primaryColor,
        },
    },

    template: (
        <view class="article">
            <wmp-markdown>{{text}}</wmp-markdown>
        </view>
    ),

    init() {
        this.contentAPI = api('content');
    },

    async onLoad({ contentID }) {
        const content = await this.contentAPI
          .cmd('getContent', { contentID })
          .call();
        this.text = content.text;
    }
});
```

- 完全使用JS编写，学习门槛低，使用一套eslint 就可做完整的静态检查
- config, style 中的变量在编译阶段替换。公共变量的定义，增强协作开发效率
- style 中出现，但在 template 中未出现的class 编译时会告警，开发人员须修复（这里未采用编译时自动去除的方式，保证源代码质量）

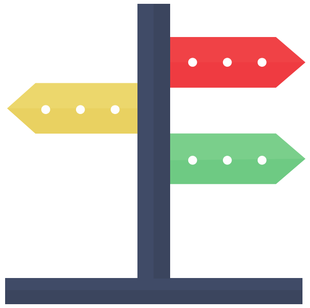
大工程编译的苦恼

整体工程编译需要6-8分钟，严重影响了开发效率

- 进行模块化拆分，每个模块安装入框架，都可运行，开发人员无需 clone、编译全部代码
- 编译速率提升了3-5倍，小程序开发工具速度飞快



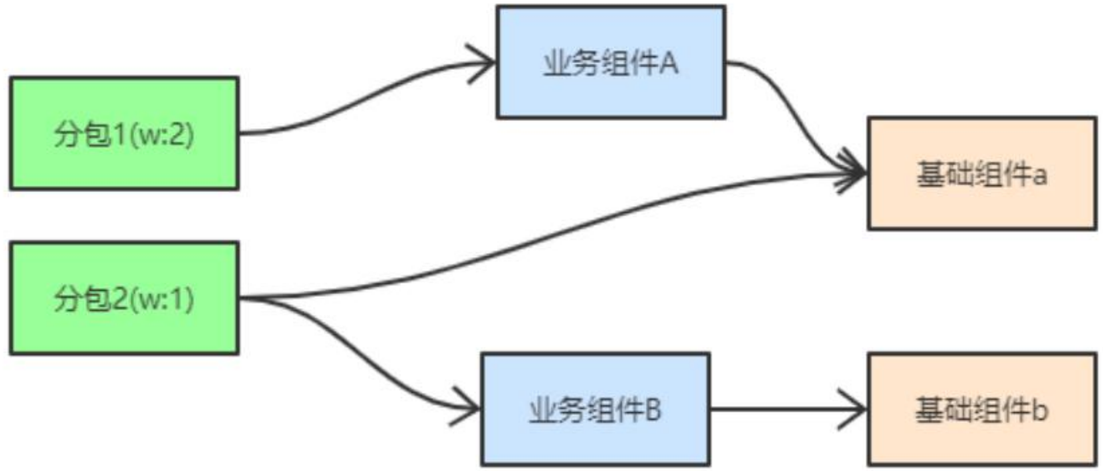
组件化能否解决包大小的问题



- 组件放主包？组件全部放主包，主包大小增加，影响较关键的首次加载
- 组件放分包？全部放分包，各分包大小集体增加，影响所有分包的加载效率

我们需要更智能的分包策略。

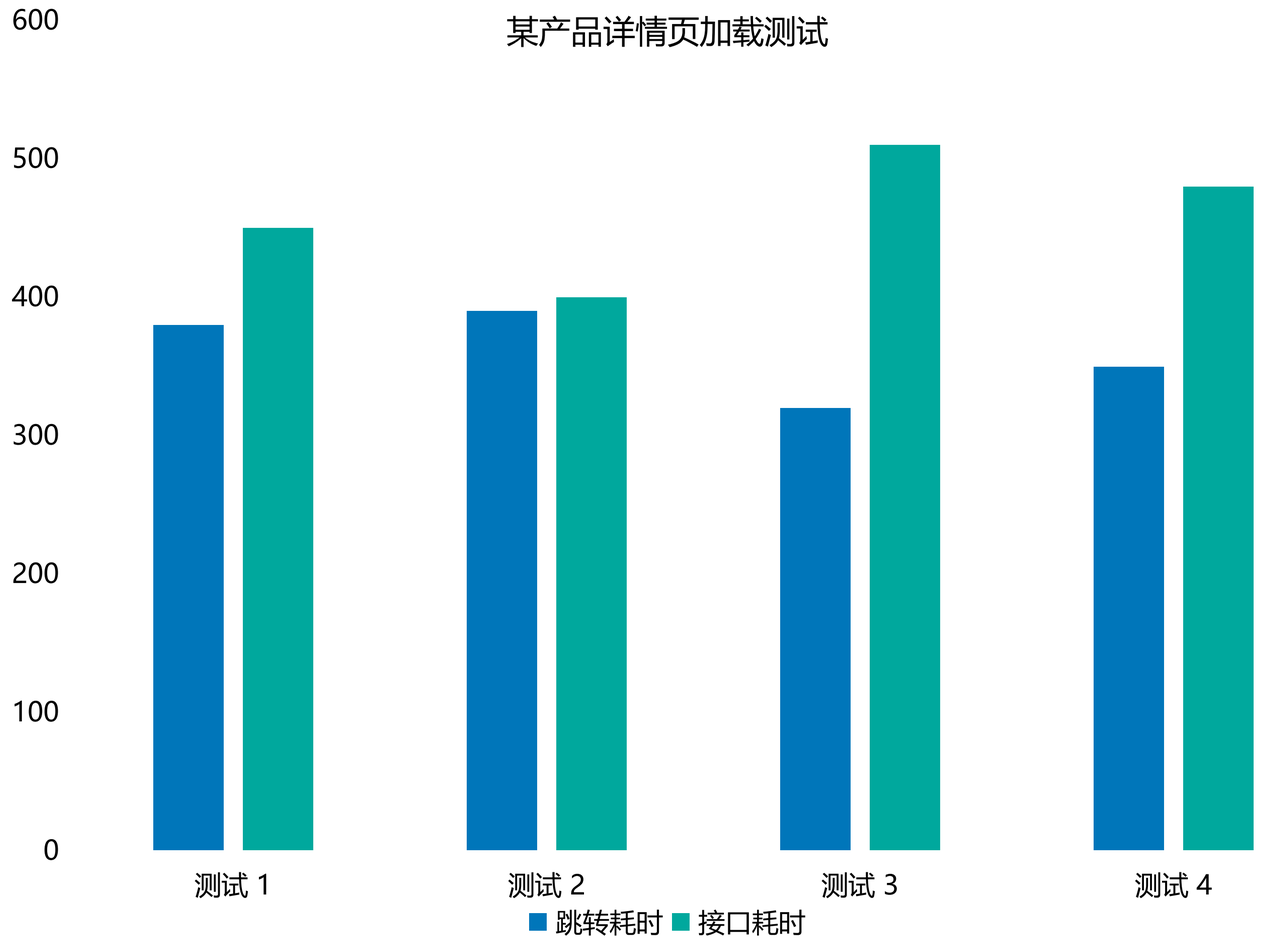
对组件做依赖分析，分析每个组件有多少个入边，大于阈值的放主包，小于阈值的各分包单独加载



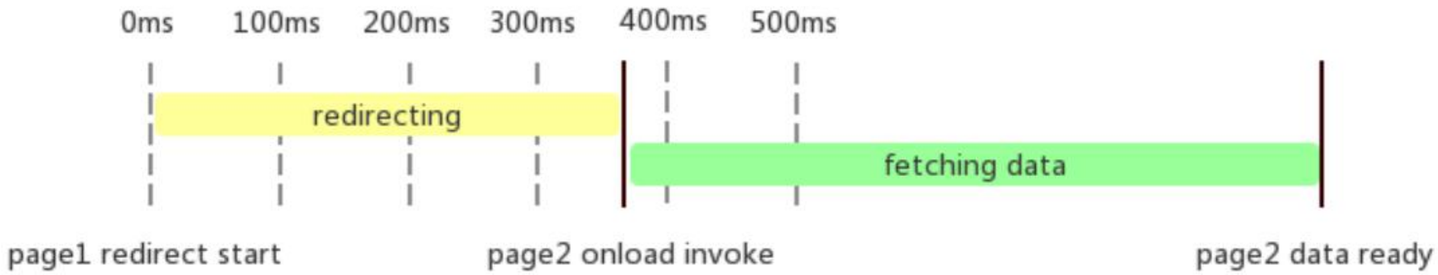
页面加载速率的分析

是什么影响了页面加载速率?

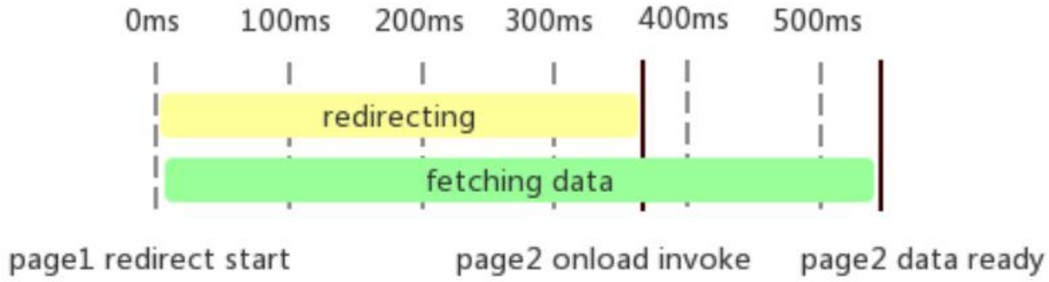
1. 页面跳转速率（200ms）
2. 数据加载速率 （200-800ms）
3. 页面渲染速率



数据预加载




```
async onLoad({ contentID }) {
    const contentAPI = api('content');
    const content = await contentAPI
        .cmd('getContent', { contentID })
        .call();
    this.text = content.text;
}
```




```
preLoad({ contentID }) {
    const contentAPI = api('content');
    return contentAPI.cmd('getContent', { contentID }).call();
},

onLoad({ contentID }) {
    this.$preLoadPromise.then(({ text }) => {
        this.text = text;
    });
}
```

总结

框架是一个团队多年经验最抽象的总结，它是工程化的基础；基于框架，我们从某种程度上解决了包大小、协同开发、编译速率、页面加载等问题

大前端架构 - API 聚合渲染

保障计划 查看详情

10万版 | 30万版 | 50万版

重大疾病保险金 (100种)	10万
特定重疾保险金 (10种)	额外给付10万
轻症疾病 (35种)	3万 + 豁免未交保费
身故保险金	返还已
投保人身故/重疾	豁免未交保费
投保年龄	0-17周岁
保险期限	至23周岁
交费期间	至23周岁

保费 ￥71/年起

微医保.少儿重疾险和微医保.重疾险的区别?

微医保.少儿重疾险保至23岁，保障白血病等100种重疾+35种轻症，其中10种少儿高发重疾更提供双倍赔付。微医保.重疾险保1年，保障100种重疾。两者可同时购买，理赔时领取两份保额。

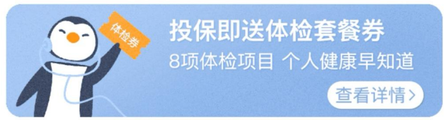
用户说 查看更多

“保费不涨价”、“保费豁免”

“这款产品重疾保障全面，人保五百强公司值得信赖。客服耐心回答问题，保费不涨价，低保费高保障，值得购买。感谢人保公司，感谢平台！”

微保用户 ★★★★★ 2019.10.23

投保即可参加以下活动



产品特色 | 理赔说明 | 我要投保

100种重疾+35种轻症，确诊即赔

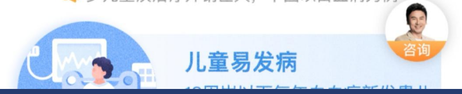
20元/月起，一顿饭钱保健康

查看详情

10种少儿特定重疾，保额翻倍

专为少儿定制，重疾最高赔付100万

少儿重疾治疗开销巨大，下图以白血病为例



理赔说明 理赔须知

理赔步骤

第1步: 拨打电话

请在保险事故发生后拨打人保寿险全国统一客服电话4008895518，我们将提供理赔指引服务。

第2步:定制方案

根据被保人的事故情况，人保寿险将定制事故查验及理赔方案。

第3步: 完成理赔

根据审核结果将理赔款支付到被保人或投保人指定账户。

常见问题 更多问题

- Q 这款产品提供了哪些保障?
- Q 什么是投保人豁免保费?
- Q 哪些重大疾病可获得双倍保障?
- Q 孩子目前有某种疾病或症状，能购买这款重疾险吗?
- Q 有了社保，还需要购买重疾险吗?

填写投保信息

1 本人信息(投保人)

姓名	*锂	已认证
身份证	420**********051	已认证
手机号	189****1307	

我已确认 投保须知及声明 | 服务协议 | 保险条款

分享 ￥20/月起 立即投保

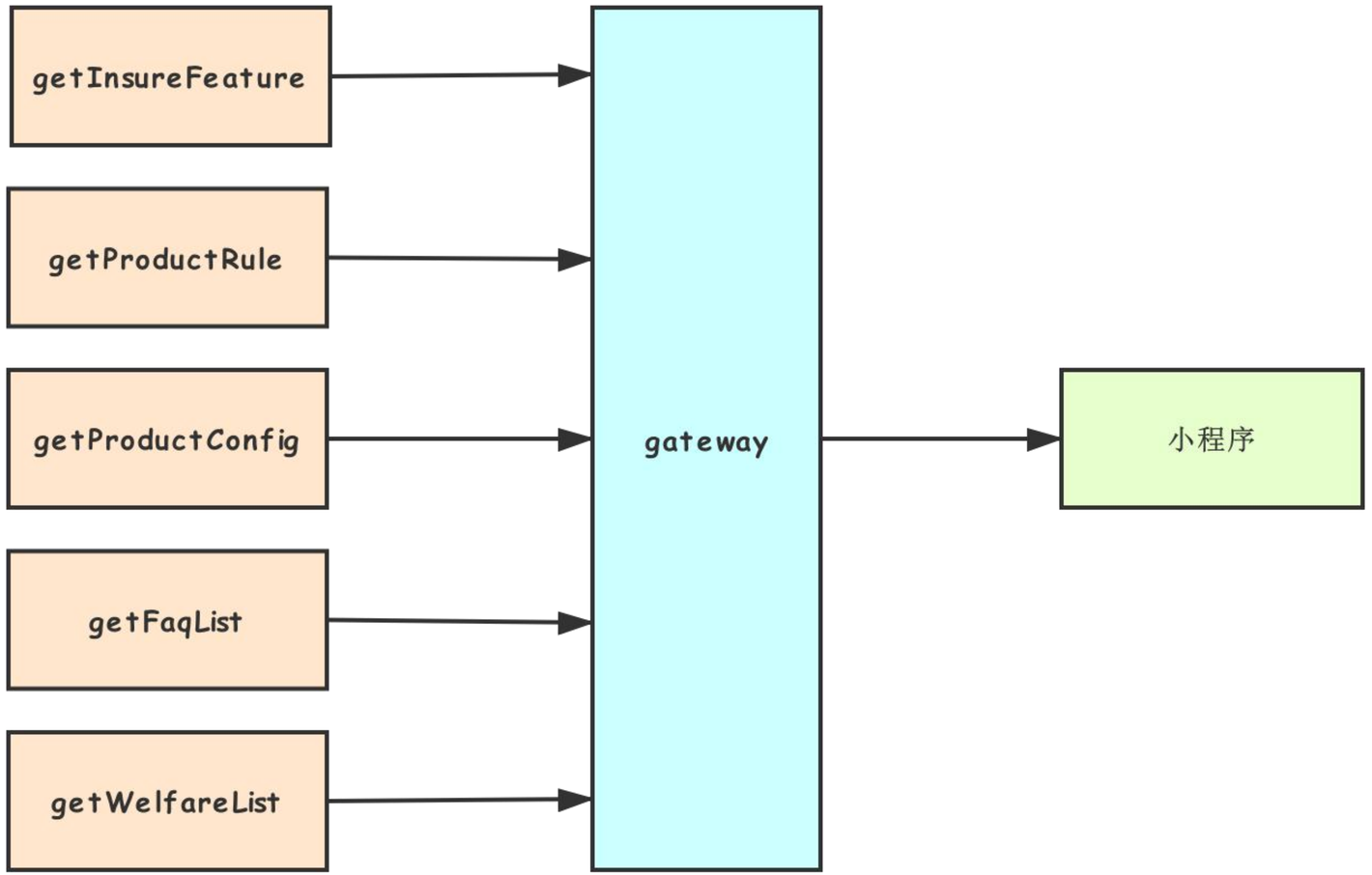
产品详情页接口

- ✓ 页面配置接口
- ✓ 方案信息接口
- ✓ 用户评论接口
- ✓ 投保福利接口
- ✓ 常见问题接口
- ✓ 投保表单接口
- ✓ 初始报价接口
- ✓ 协议列表接口

带来的问题

1. 页面并发请求过多，页面加载缓慢
2. 业务逻辑复杂，小程序包大小增加
3. 改动视觉呈现，需小程序发版支持

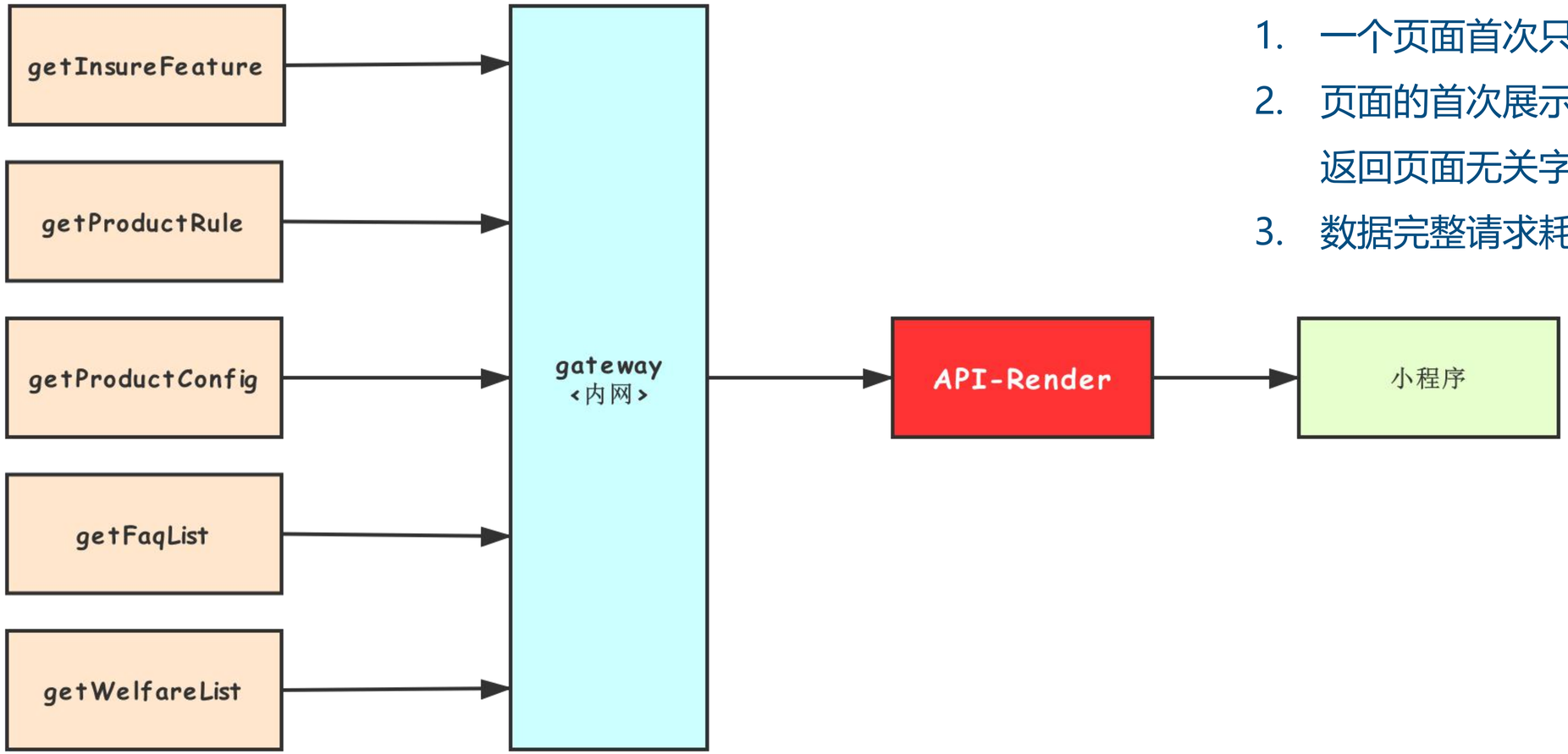
以前的调用方式



小程序通过网关直接请求业务服务

1. 一个页面并行加载过多请求
2. 业务服务接口返回的数据，大部分小程序不关心
3. 数据组装、判断和渲染逻辑全部放在小程序内
4. 完整请求产品详情页接口耗时1.5s左右

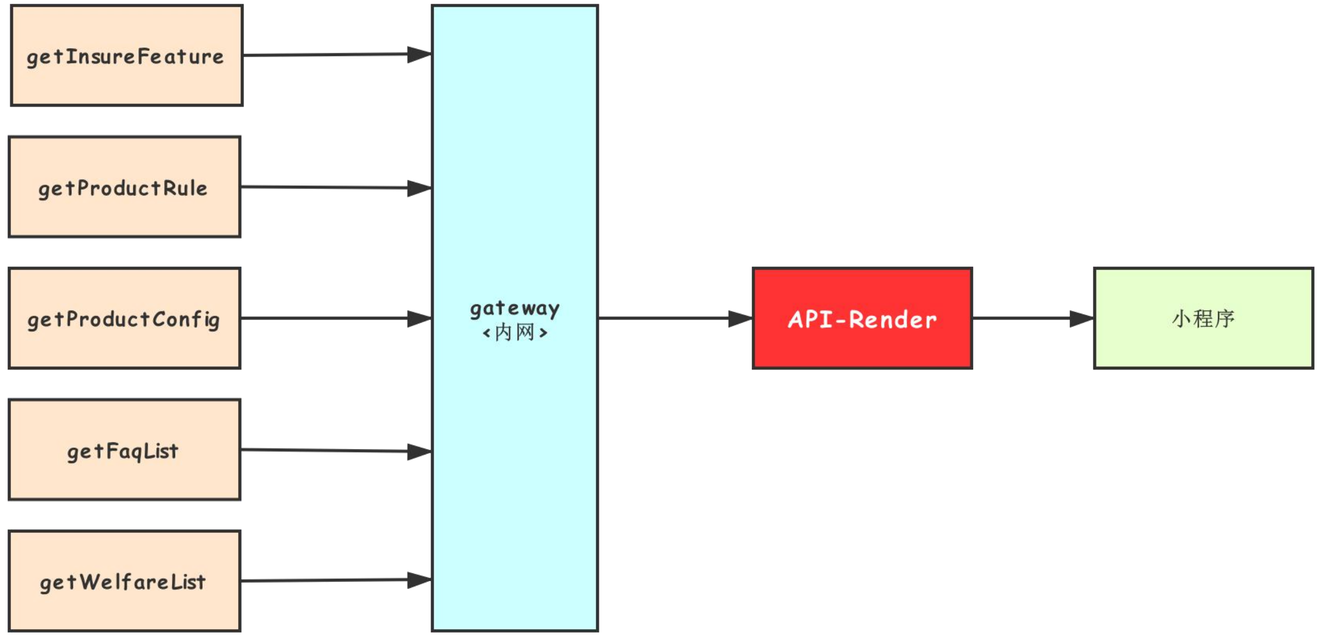
改进后的调用方式



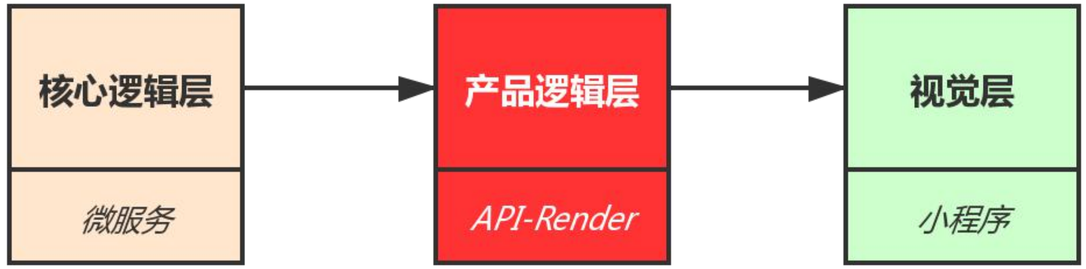
小程序和 API-Render 交互

1. 一个页面首次只加载一个请求
2. 页面的首次展示逻辑放到 API-Render，不返回页面无关字段
3. 数据完整请求耗时下降为 0.5s 左右

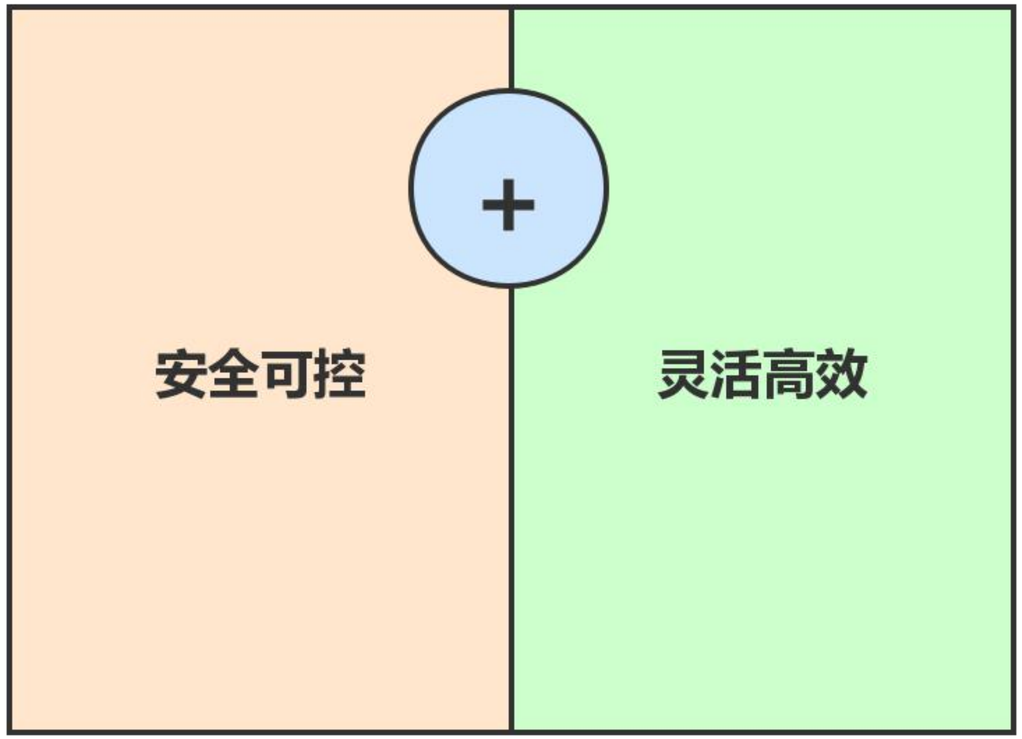
灵活与稳定的分层设计



1. **稳定**的保险核心接口，面向稳定的保险逻辑
2. **较稳定**的小程序，面向规范的视觉元素设计
3. **灵活**的 API-Render，面向复杂多变的产品逻辑

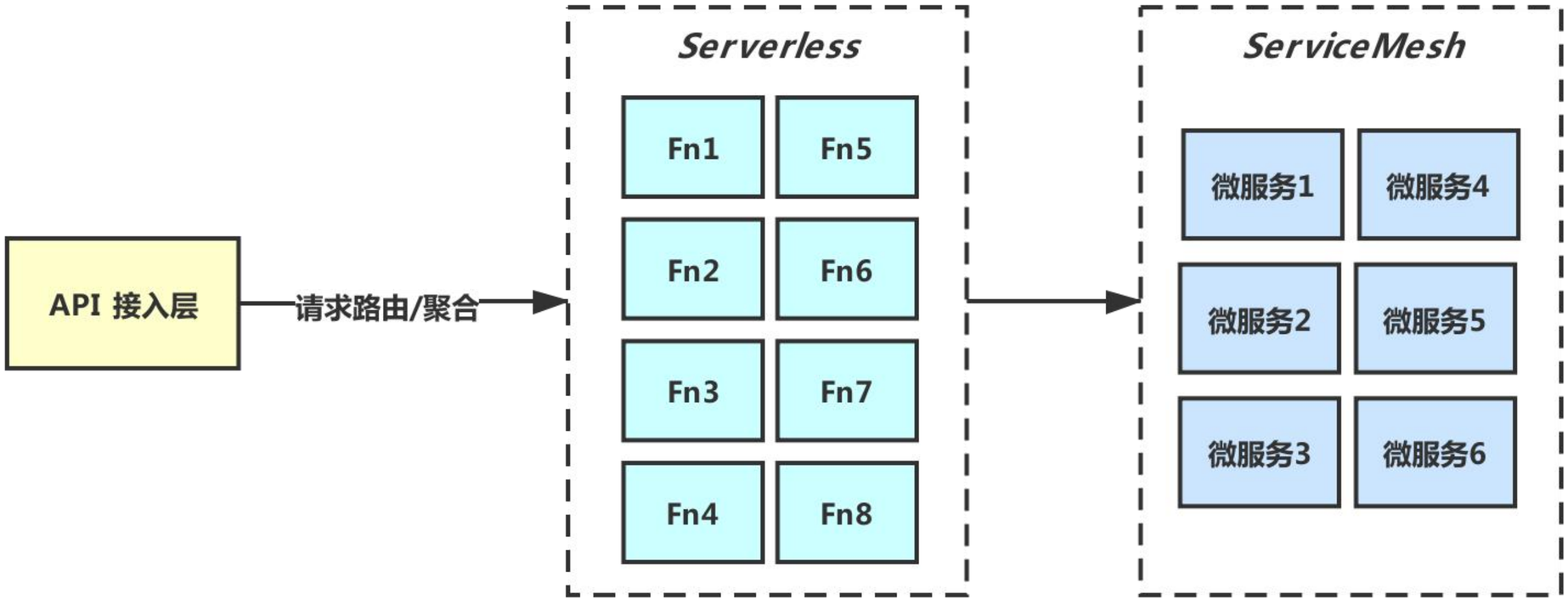


如何保证产品逻辑层灵活安全的发版?



更小粒度的发布

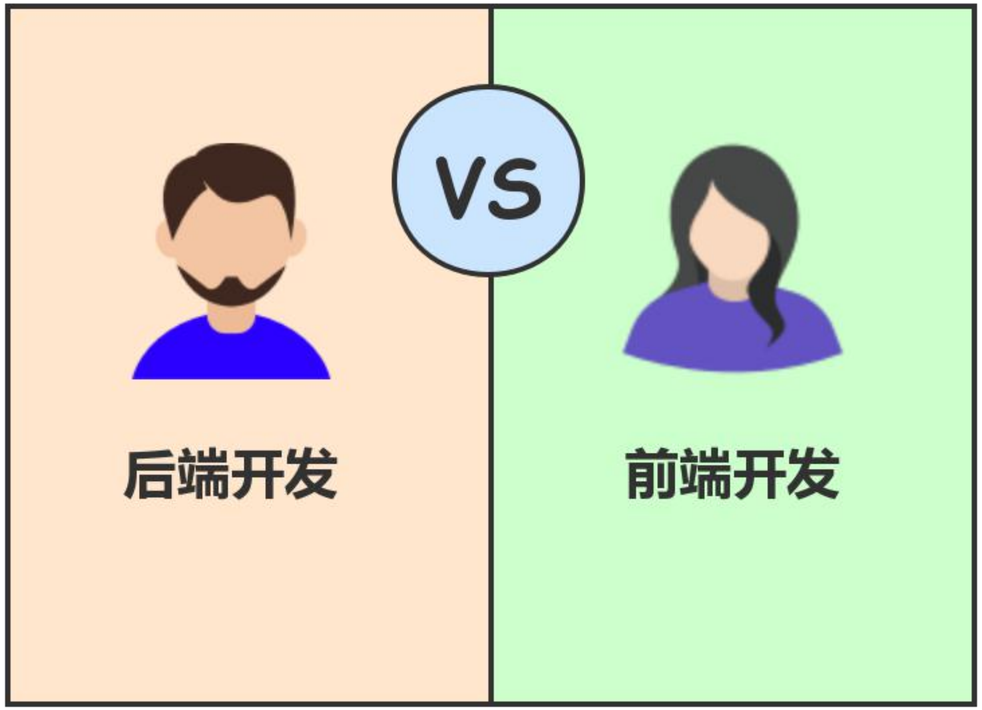
保证 API-Render 灵活性



使用 Serverless 部署

1. 以函数为粒度发布，可做到随改随发
2. 以函数为粒度分配资源，高频和低频接口的资源分配更加合理(产品详情页 vs 提交订单)

产品逻辑层是前端开发还是后端开发?



因为需求由前端驱动开发

前端开发承担这项任务，对前端开发人员的技能提出了更高的要求

简要设计文档 + CodeReview 方式

总结

API-Render 作为非常重要的产品逻辑层，很好的隔离核心逻辑和视觉逻辑，

让各层分工更加明晰专注

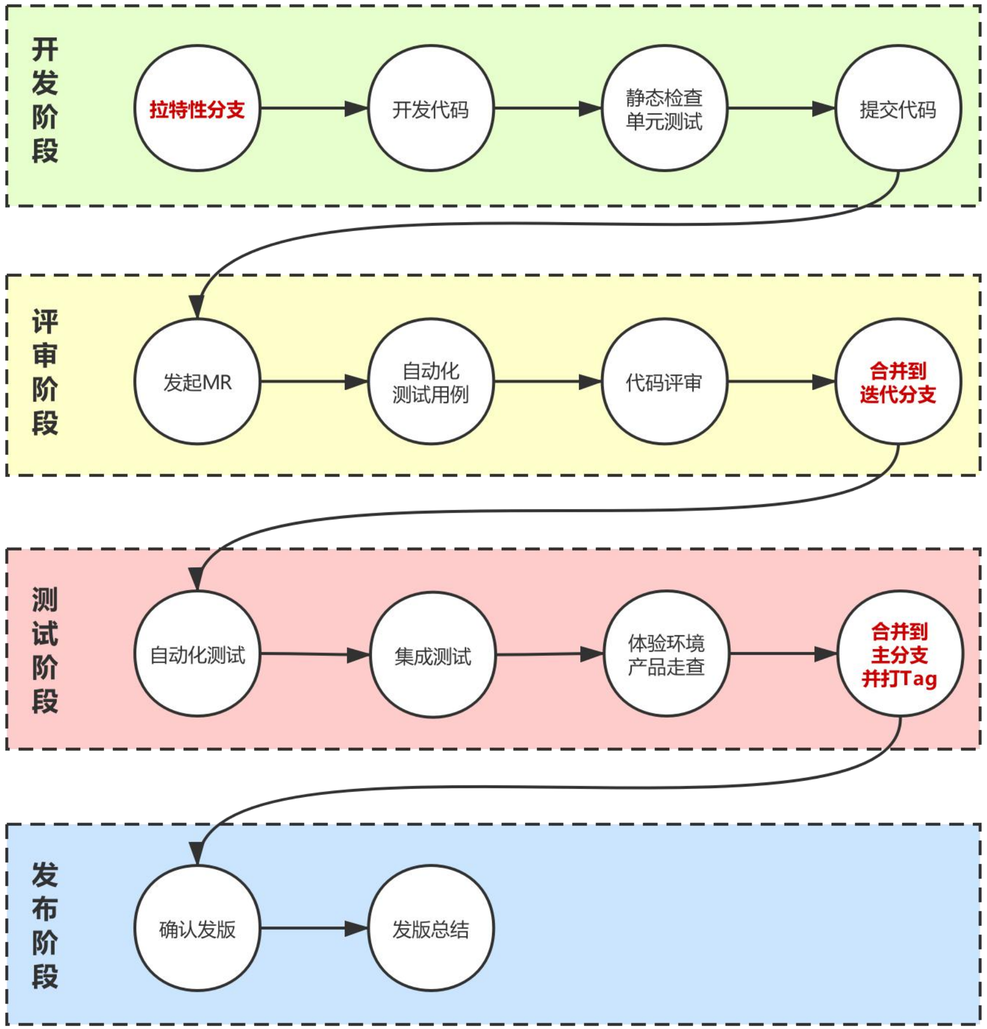
Serverless的引入让产品逻辑层的发布和部署更加灵活可控

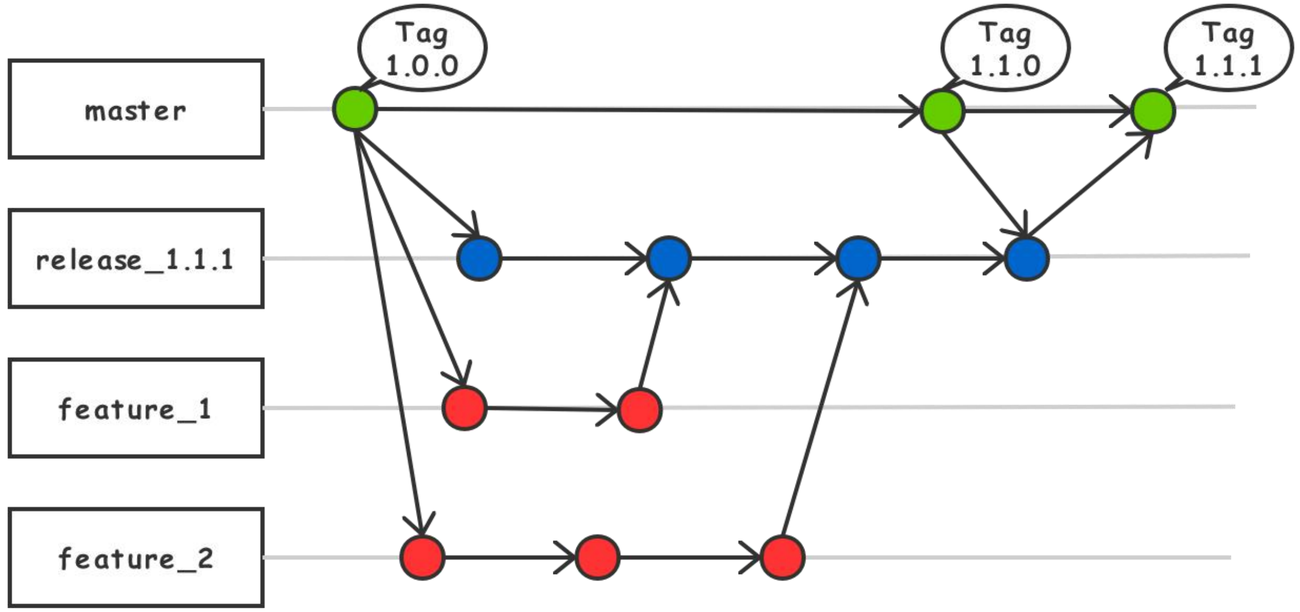
迭代开发利器 - 持续集成系统

存在的问题

1. 小程序必须作为一个整体发布
2. 一个迭代，8 个产品条线同时开发，涉及 180+ 个需求
3. 为了较大限度的避免代码冲突，目前拆成了 68 个 Git 仓库
4. 单周迭代，双周迭代，紧急特性

迭代黄金流程 & Git 分支规范

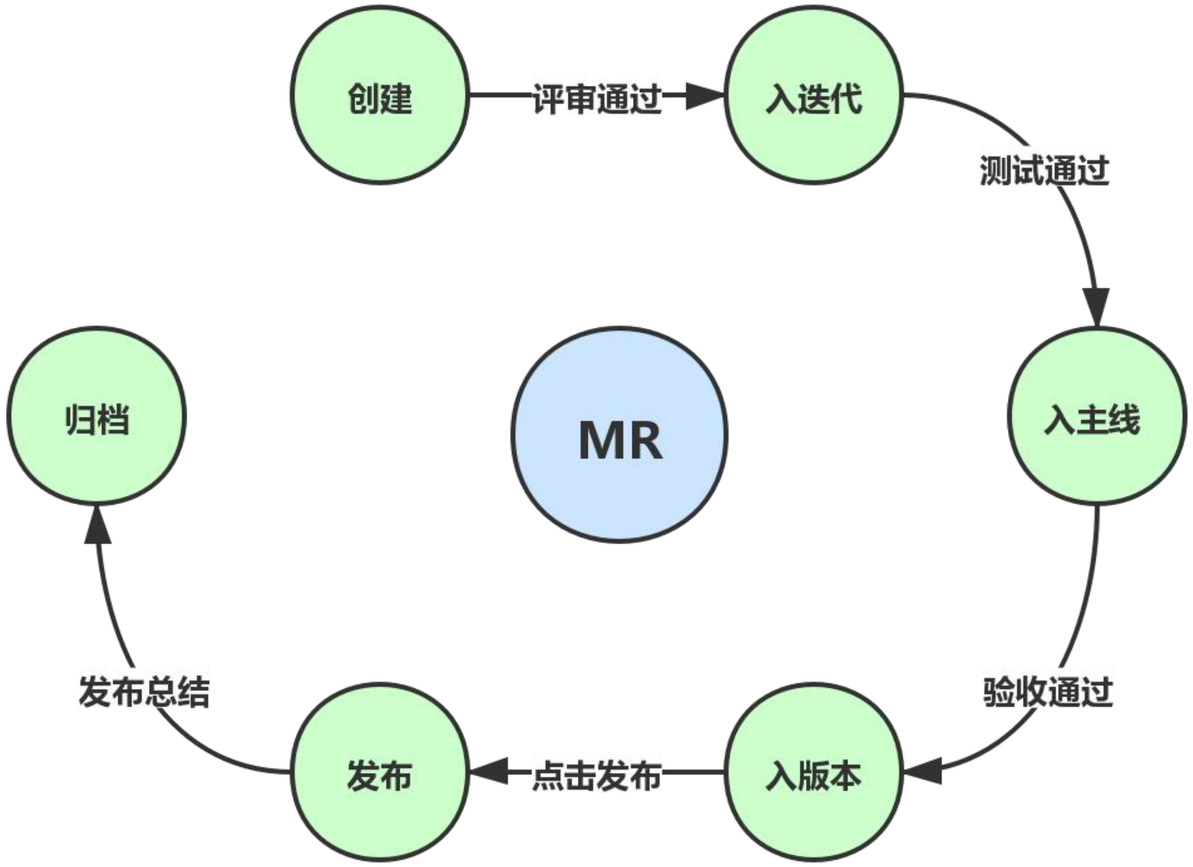




1. master 为总体基线
2. release 为迭代基线
3. feature 为特性开发分支
4. tag 用于发布

如何管理如此复杂的流程?

MR - 流程的核心驱动力



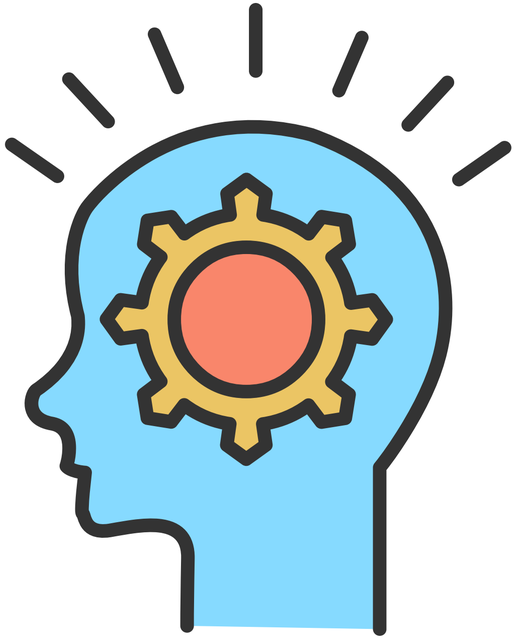
MR 的生命周期

对复杂迭代流程的管理其实就是对若干MR生命周期的管理

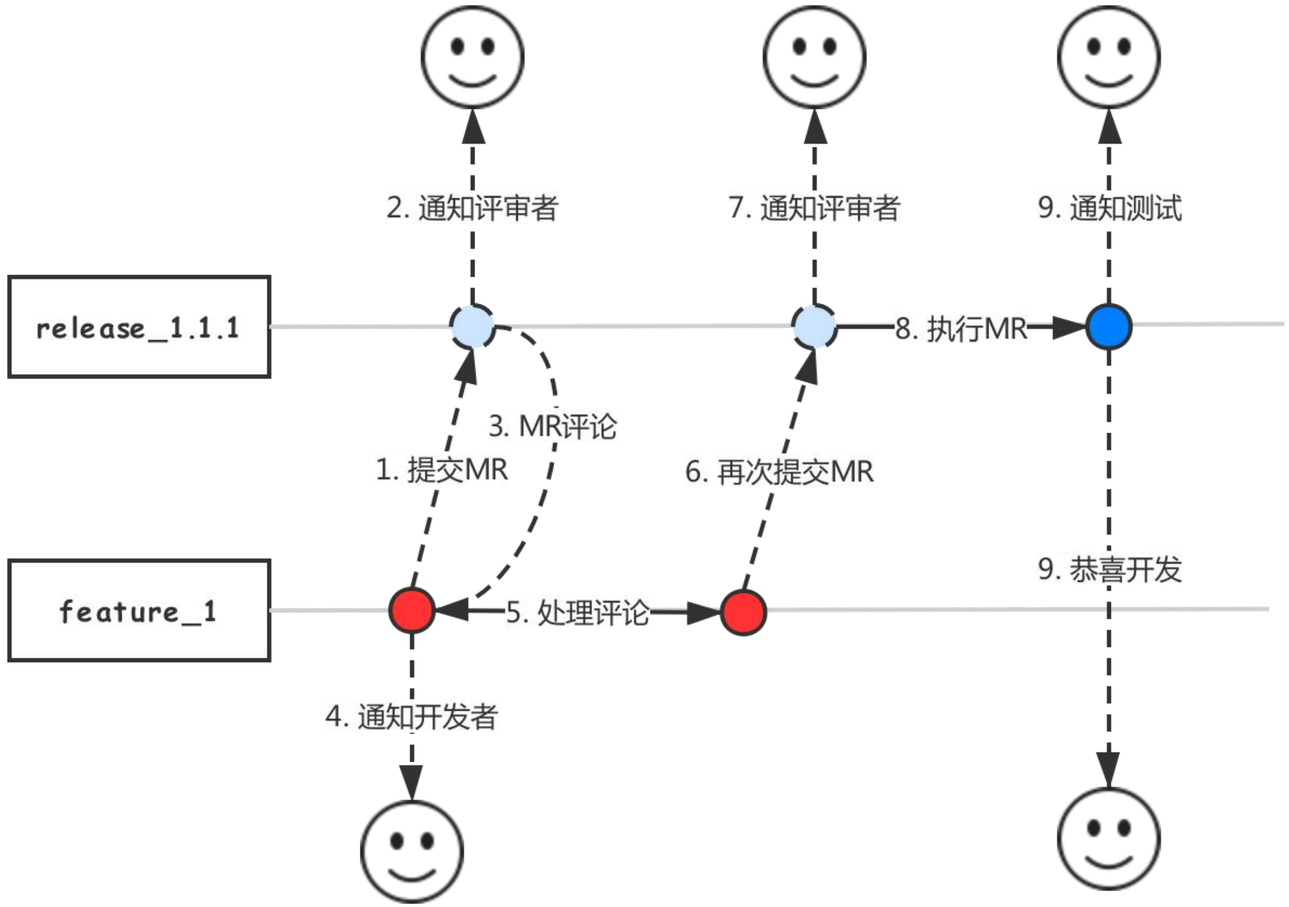
思考：如何实施MR的质量管理?

MR质量保证

1. 为MR编写有效的描述 ★★
2. 静态检查 ★★
3. 单元测试 ★★★★★
4. 代码评审 ★★
5. 自动化测试 ★★★★
6. 集成测试 ★★★
7. 产品验收 ★★
8. 线上监控 ★★★★
9. 有效复盘 ★★★



CodeReview 的引入



CR 过程最重要的是什么?

MR 处理的及时性

【AT-config/config-portal】代码评审

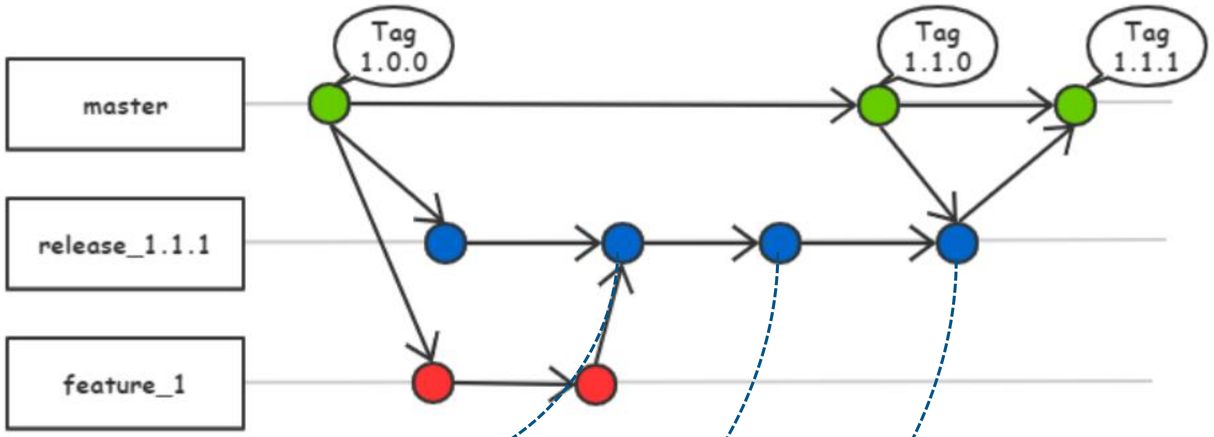
ed 发起了代码评审，请及时处理

（第1次通知）

MR 处理的及时性是影响 CR 效率的最重要因素

详情

持续集成工具的引入



release_2.21.0

开始测试

分支名	描述	状态	测试人员	开发人员
feature_auto_renewal_ui	自动续保组件UI调整	已构建	-	[illegible]
hotfix_papay_report	签约日志上报	测试通过	[illegible]	[illegible]
feature_finance_notice	补充财务告知	测试不通过	[illegible]	[illegible]
merge_2.20.1	merge 2.20.1	-	-	-

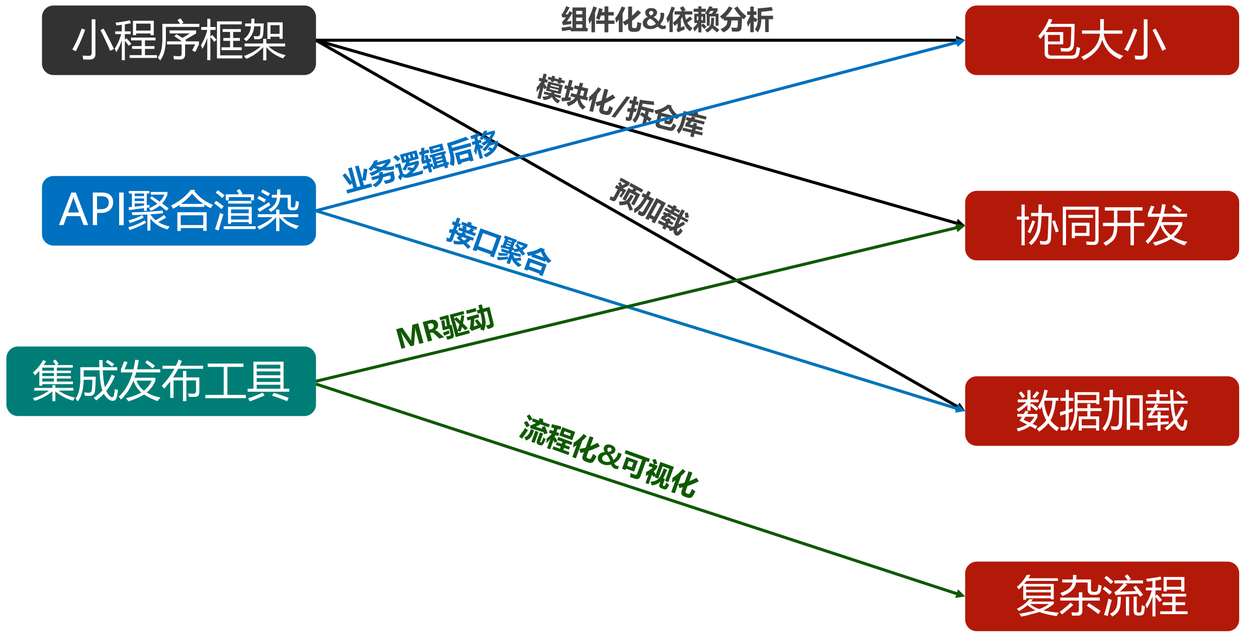
总结

借助工具的自动化，可视化， 保障了复杂的迭代开发流程有条不紊的进行。

MR 会作为各迭代的重要资产沉淀下来，开发人员、评审人员会严肃对待，
MR的描述越写越完善，提测的质量也有明显好转。

总结与展望

连线题


小程序框架
API聚合渲染
集成发布工具
包大小
协同开发
数据加载
复杂流程
组件化&依赖分析
模块化/拆仓库
预加载
业务逻辑后移
接口聚合
MR驱动
流程化&可视化

THANKS

GMTC
全球大前端技术大会