前端技术规划
#3

- #3
- #2
- #1

前端技术

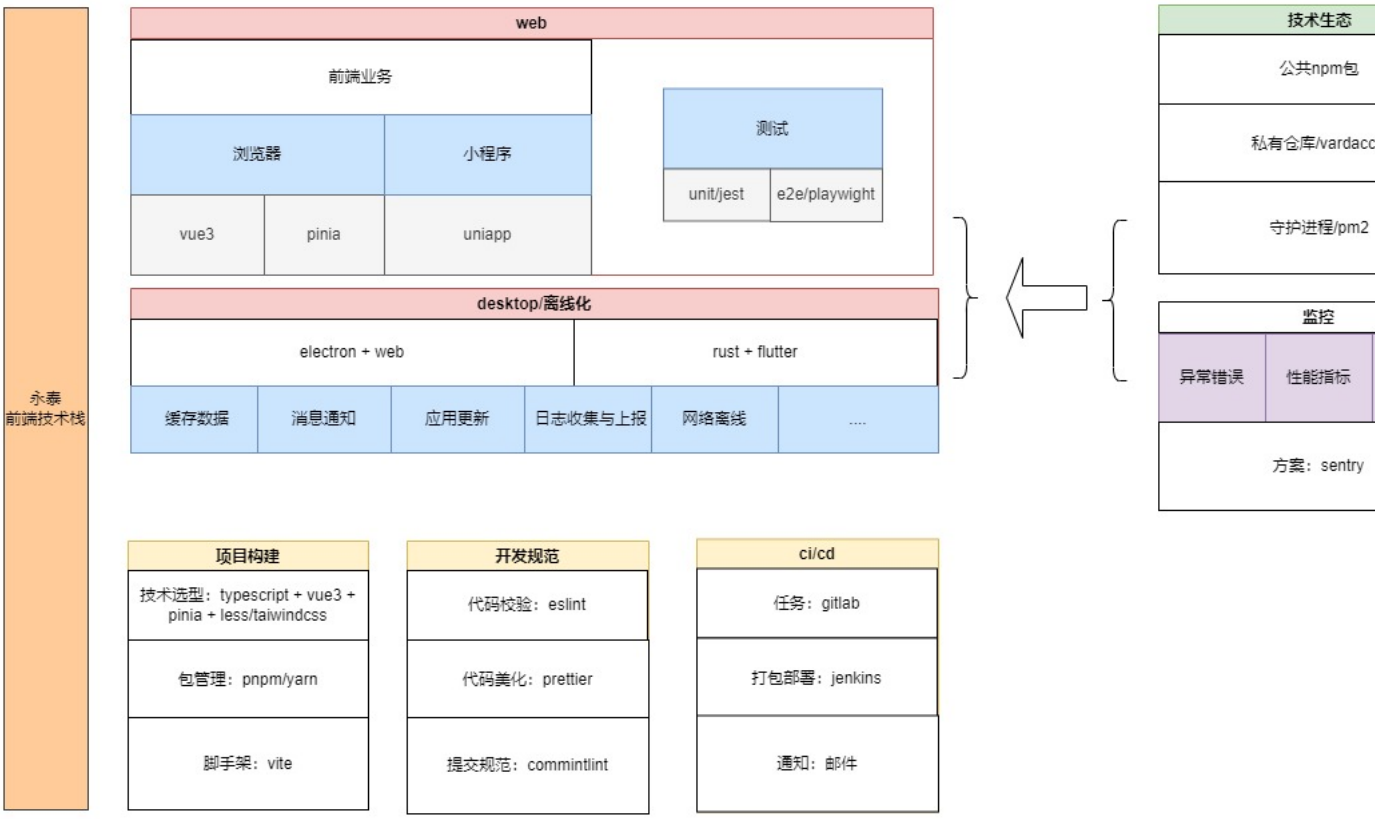
背景

在发展新项目之初，为了能更好的统一前端技术选型，更好的规划前端整个架构，以便能让不同新的项目能够高效，快速的搭建，输出一套统一的技术架构是很有必要的。

该套技术架构不仅需要统一了前端的技术栈，而且还需要引入持续集成和持续交付的能力（cicd）。

有了正常的迭代之后，还得保证项目的稳定性，提升项目的性能，因此还需引入比较完善的前端监控体系。

前端架构规划

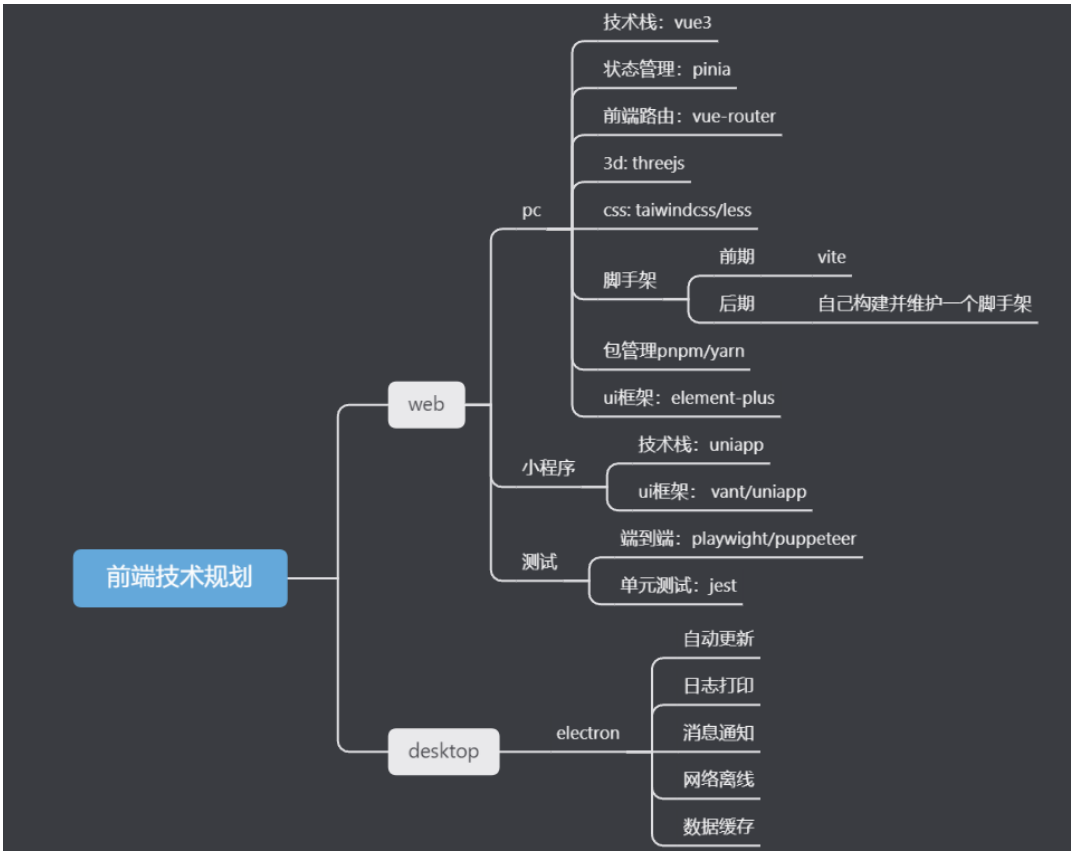


业务技术选型

市面上比较成熟的前端框架就是react, vue, angular，由于国内比较偏向vue和react居多，所以从这俩个中我们选择比较适合公司业务的技术栈。

vue，react都能轻松应对复杂的项目，由于vue3也出了composition-api，相对大型项目来说也能驾驭得住。e3 + typescript，配合volar也可以做到健全的类型推断，且vue3的周边生态比较完善，基于现有的业务场景，更建议采用vue3。

所以前期在技术选型上，借助社区提供的脚手架vite，搭建一套vue3的生态。



web浏览器端

技术选型

web采用主流的技术配套：vue3+ pinia + vue-router + typescript + less/taiwindcss。

其中数据状态更推荐用pinia，而不是vuex，pinia采用函数式的写法更契合函数式的组件。

小程序采用主流的技术配套：uniapp(vue3)。

小程序可以依赖第三方的ui框架，比如vant, 或者使用内置的ui。

项目搭建

对于项目创建，前期推荐采用社区比较采纳的方案：vite。

后期应该有一套健全的包管理系统，可以自己定制脚手架，新创建的项目都依赖该脚手架来创建，这样项目的配置能达到统一。

pc应用端（离线方案）

pc离线方案采用electron，尽管electron有各种问题，比如说包体积过大，crash问题，以cpu内存占用过高的问题，但是electron对于当下的情况，在开发人员不足情况下，是能做到1 + 1> 2的。

采用离线方案，发布客户端app，也会增加相应的业务问题，比如自动更新，数据缓存，数据加密，收集日志，网络离线等。

前端工作流

统一代码规范

由于项目会越来越大，项目的复杂度越来越高，并且可能会有新同事的加入，这时需要通过插件工具，在代码开发和提交阶段，校验并美化代码。

prettier： 对代码风格进行统一的美化

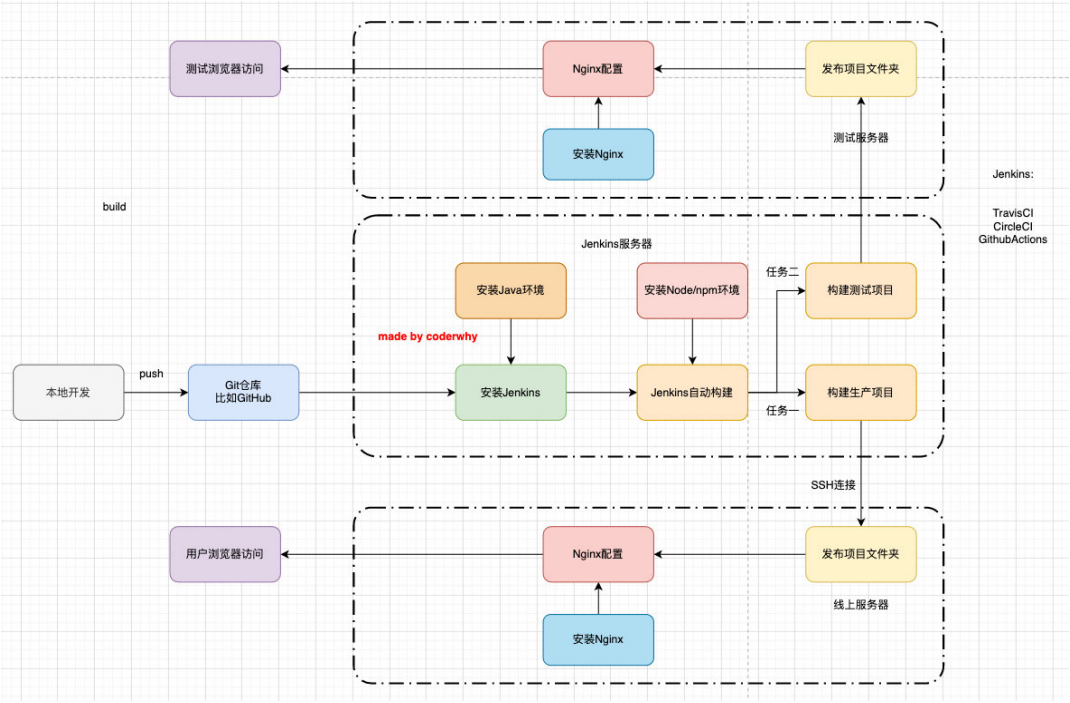
eslint：代码的书写风格规范也必须保持统一（比如禁止使用var，字符串用单引号）。

commitlint：对git提交规范进行校验

持续集成与交付

前端的需要自动化的构建并部署上线（测试环境，正式环境）。这里分为三个具体的环节：

1. 构建task: 需要通过编写gitlab的task，让任务在merge完代码之后自动执行构建脚本。
2. 构建jenkins：通过编写jenkins的配置，执行打包脚本，部署到服务器
3. 通知开发：可以通过插件，在取得部署成功之后的消息，发送邮件告知开发者。



前端基础建设

私有仓库

随着项目的越来越大，肯定会出现需要封装私有组件或插件的情况，所以有必要自己构建一套前端的私有仓库，可以用业界成熟的方案verdaccio + pm2。

这里对于包管理强烈建议使用pnpm，因为pnpm的包管理方式会比较优秀。

前端监控系统

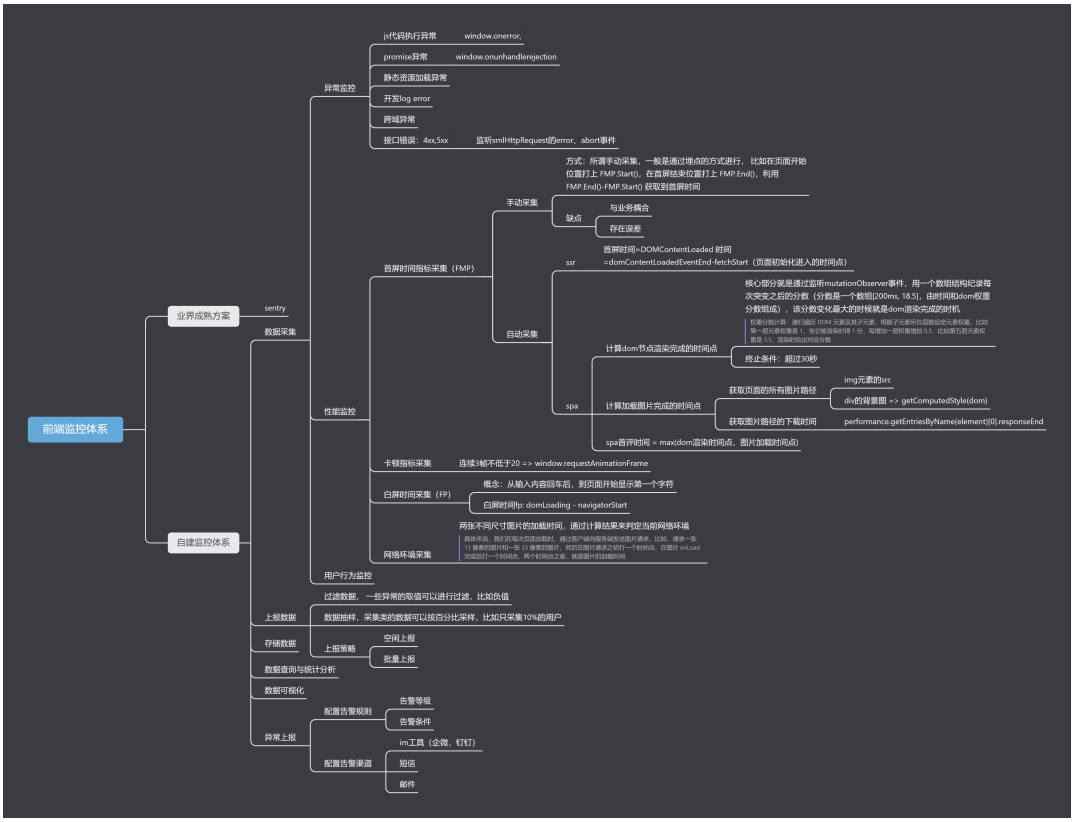
为了能让前端系统更加健壮，提高网站的访问速度，有必要集成前端的监控系统。

前端监控系统可以优化以下三项指标：

- 异常错误：由于线上代码可能出现异常，我们可以通过监控系统及时发现问题，通过sourseMap定位代码位置，并可以快速打补丁修复。
- 性能指标：前端在业务代码中可以做一些性能埋点，在监控系统中以一种可视化的方式展现出来，方便分析网站的各项指标，比如白屏时间（fp），最大内容绘制时间（LCP），首次输入延迟（FID），网络环境等。
- 用户行为：监控系统还能记录用户的行为，方便还原“线上事故”现场。

社区方案：sentry是一个成熟的方案，有开源也有商业化部分，前期项目落地可以用开源版本。

后续方案：可以自己搭建一个监控系统。


前端监控体系
业界成熟方案
sentry
自建监控体系
数据采集
异常监控
js代码执行异常 window.onerror,
promise异常 window.onunhandlerejection
静态资源加载异常
开发log error
跨域异常
接口错误：4xx,5xx 监听xmlHttpRequest的error、abort事件
性能监控
首屏时间指标采集（FMP）
手动采集
方式：所谓手动采集，一般是通过埋点的方式进行，比如在页面开始位置打上 FMP.Start()，在首屏结束位置打上 FMP.End()，利用FMP.End()-FMP.Start() 获取到首屏时间
缺点
与业务耦合
存在误差
自动采集
首屏时间=DOMContentLoaded 时间
ssr =domContentLoadedEventEnd-fetchStart（页面初始化进入的时间点）
spa
计算dom节点渲染完成的时间点
核心部分就是通过监听mutationObserver事件，用一个数组结构纪录每次突变之后的分数（分数是一个数组[200ms, 18.5]，由时间和dom权重分数组成），该分数变化最大的时候就是dom渲染完成的时机
权重分数计算：遍历遍历 DOM 元素及其子元素，根据子元素所在层数设定元素权重，比如第一层元素权重是 1，每记录深层时得 1 分，每增加一层权重增加 1.5，比如第五层元素权重是 3.5，当等时给出时间分数
终止条件：超过30秒
计算加载图片完成的时间点
获取页面的所有图片路径
img元素的src
div的背景图 => getComputedStyle(dom)
获取图片路径的下载时间 performance.getEntriesByName(element)[0].responseEnd
spa首评时间 = max(dom渲染时间点，图片加载时间点)
卡顿指标采集 连续3帧不低于20 => window.requestAnimationFrame
白屏时间采集（FP）
概念：从输入内容回车后，到页面开始显示第一个字符
白屏时间fp: domLoading - navigatorStart
网络环境采集
两张不同尺寸图片的加载时间，通过计算结果来判定当前网络环境
网络环境采集：我们可写次页面加载时，通过客户端向服务端发送图片请求，比如，请求一张 11 像素的图片和一张 33 像素的图片，然后在图片请求之初打一个时间点，在图片 onLoad 完成后打一个时间点，两个时间点之差，就是图片的加载时间
用户行为监控
存储数据
上报策略
空闲上报
批量上报
数据查询与统计分析
数据可视化
异常上报
配置告警规则
告警等级
告警条件
配置告警渠道
im工具（企微，钉钉）
短信
邮件
上报数据
过滤数据，一些异常的取值可以进行过滤，比如负值
数据抽样，采集类的数据可以按百分比采样，比如只采集10%的用户