公司

和生创新技术

基础技术与规范

标签语义化、布局、es6

- **HTML5 的语义化标签（如 `<article>` `<section>`）在电商网站中如何应用？相比 div 有什么优势？**
  - 比如商品详情页，用 `<article>` 包裹商品主体信息（标题、价格、描述），`<section>` 拆分规格选择区、评价区等模块，`<nav>` 定义面包屑导航，`<aside>` 放相关推荐商品；
  - 相比div，一是提升 SEO，搜索引擎更容易识别商品核心内容；二是代码可读性更高，新同事接手时能快速定位模块结构；三是增强可访问性，屏幕阅读器能按标签层级解析内容，适配残障用户；
- **用 CSS 实现一个商品列表的响应式布局（PC 端 3 列、移动端 1 列），你会选择 Flex 还是 Grid？为什么？**
- **ES6 的 Promise 和 async/await 在处理电商场景中的异步请求（如加入购物车后同步库存）时，你是如何避免回调地狱的？举个项目中的例子。**
  - 比如 ' 加入购物车→更新库存→同步本地缓存 ' 的流程，使用async/await，让代码逻辑更线性，有助于逻辑理解和排查。

框架与工具实战

react、next.js、shopify、工程化

- **在 GogoalShop 中，你用 Redux 管理了哪些状态？有没有遇到过状态冗余的问题？如何优化的？**
  - GogoalShop 中，Redux 主要管理三类全局状态：用户信息（登录态、地址）、购物车（商品列表、选中状态）、全局配置（汇率、语言）；
  - 初期确实遇到过状态冗余问题 —— 比如商品详情页和购物车都存了商品价格，导致同步不一致；
  - 用 Redux Toolkit 的 `createEntityAdapter` 规范化商品数据（以 ID 为 key 存储，避免重复）；
  - 拆分切片（slice）：将购物车、用户、配置拆分为独立切片，各自维护 reducer 和 action；
  - 用 `createSelector` 做派生状态计算（比如购物车总价 =∑(商品单价 × 数量)，避免重复计算）；
    改造后，Redux DevTools 调试时能清晰追踪状态变更来源。
- **解释下 Next.js 的 SSR、SSG、ISR 的区别？在你的电商项目中，商品详情页用了哪种渲染方式？为什么？**
  - 先说区别：
    - **SSR（Server-Side Rendering，服务端渲染）**，每次用户请求时，服务器都会动态生成完整的 HTML 页面并返回给客户端，客户端接收后直接渲染。内容**实时性**极高，但**服务器压力**较大，**首屏加载速度**受服务器响应时间影响。
    - **SSG（Static Site Generation，静态站点生成）**，在项目构建（`next build`）时，提前预渲染好所有页面的 HTML 并存储在服务器，用户请求时直接返回静态 HTML，无需动态生成。性能极佳（静态资源可缓存，加载速度快），服务器压力小，但**内容更新不及**时（需重新构建才能更新页面，适合内容长期不变的场景）。

    - **ISR（Incremental Static Regeneration，增量静态再生）**，结合 SSR 和 SSG 的优势，先在构建时预渲染静态 HTML，之后通过配置 `revalidate` 参数，让页面在指定时间间隔或按需重新生成（无需全量重构），**兼顾性能（静态资源缓存）和实时性（定期 / 按需更新）**，服务器压力小，适合内容有一定更新频率但非实时变动的场景。
  - 所以，商品详情页我们采用的是 ISR：
    - **商品信息**（价格、描述）不会实时变更（每天同步一次），不需要 SSR 的实时性；
    - 但**商品库存**可能频繁变动，可让页面在 1 小时后自动重新生成，既保证内容新鲜度，又避免 SSR 带来的服务器压力。
    - 对于**热销或秒杀商品**，revalidate配置缩短至5-10min，根据业务调整。
  - 综上，对于这三种渲染方式的选取，其实就是权衡**内容新鲜度**和**服务器压力**。
- **Next15 迁移时，最大的挑战是什么？比如旧代码的 TS 类型适配问题，你是如何解决的？迁移后开发效率提升了多少？**
  - 先回答最大挑战：集中在 **React Server Components (RSC) 模型的全面适配** 以及 **TypeScript 类型系统的兼容调整**
    - 旧项目（基于 Next.js 13）混合了 Page Router（客户端为主）和早期 App Router 的写法
    - **组件类型模糊**：旧代码中未明确区分 “客户端组件” 和 “服务器组件”，导致迁移后频繁出现 “服务器组件使用浏览器 API”“客户端组件缺少 `'use client'` 声明” 等 TS 类型报错
    - **数据获取逻辑不兼容**：旧代码依赖 `getServerSideProps` / `getStaticProps`（Page Router），而 Next.js 15 推荐在服务器组件中直接用 `fetch` 或 Server Actions，导致数据类型定义（如 `Props` 类型）与新数据流不匹配，TS 类型推断失效。
    - **第三方库类型冲突**：部分依赖（如旧版 Redux、UI 组件库）未适配 RSC 类型，在服务器组件中引入时 TS 会提示 “客户端依赖不能在服务器组件中使用”
  - 我采取策略是：**渐进式分步迁移**（先迁14，再迁15顺便集成TS），将迁移事项告知组员，严格审查组员代码
    - **路由层：新旧路由共存，从非核心页面开始迁移**
      - 先迁移非核心页面（如帮助中心、关于我们等静态内容页）到 `app/` 目录，这些页面依赖少、改动风险低，适合验证 RSC 模型的适配逻辑（如服务器组件与客户端组件的拆分）。
      - 核心业务页面（如首页、详情页、购物车、结算页）仍保留在 `pages/` 目录，通过 Next.js 的**路由优先级机制**（`app/` 目录优先于 `pages/`）保证新旧路由兼容，避免影响用户下单等核心流程。
      - 待非核心页面迁移稳定后（约 1 周），再逐步将核心页面按 “用户访问频率从低到高” 排序迁移，每迁移一个页面就简单回归测试下单流程，确保无异常后再进行下一个页面迁移。
    - **组件层：先 “标记分类” 再 “改造迁移”，避免批量重构风险**
      - 第一步：批量扫描旧组件，通过 `'use client'` 声明和 TS 类型标记（如 `ClientComponent` 别名）区分组件类型，仅修改 “明显违规” 的组件（如服务器组件中调用 `window` 的代码），确保现有功能不崩。
      - 第二部：对标记为 “客户端组件” 的模块，逐步重构逻辑（如将数据请求从客户端 `useEffect` 移到服务器组件的 `fetch`），优先改造 “性能敏感型组件”（如商品卡片列表），非关键组件（如 footer、导航栏）延后处理。

    - 过程中保留旧组件的 “兼容版本”，通过条件导入（`import dynamic from 'next/dynamic'`）在新旧路由中复用，避免重复开发。
  - **数据层：中间层兼容新旧数据逻辑，平滑过渡数据流**
    - 封装统一的数据请求中间层（`data-fetchers/`），同时支持旧模式（`getServerSideProps` 调用）和新模式（服务器组件 `fetch`），内部通过环境变量控制切换逻辑（如 `NEXT_PUBLIC_USE_NEW_FETCH`）。
    - 先在非核心页面启用新数据逻辑，验证数据类型适配（如旧 `Props` 类型与新 `ProductData` 的转换），待类型稳定后，再在核心页面逐步替换，期间通过日志监控数据一致性。
  - **TS渐进式集成**：用 `allowJs: true` 渐进式迁移，对于核心页面，先// @ts-ignore跳过，逐步补全类型（从接口返回值开始）
  - **发布策略：小步迭代**：迁移过程中保持 “每周一个小版本” 的节奏，每个版本仅包含 1-2 个页面
- 你开发的 TT-Options 应用是如何实现商品自定义印号功能的？和 Shopify 的商品 API 是如何交互的？了解 Liquid 模板吗？如果要改 Shopify 主题的商品页，你会从哪里入手？
- 你自定义过 Webpack 的 Loader/Plugin，能举个具体场景吗？比如实现了什么功能，解决了什么问题？

电商项目与性能优化

性能优化、项目

- 你说 GogoalShop 首屏加载减少 3S，具体用了哪些手段？比如图片优化用了 WebP 还是懒加载？代码分割是按路由还是组件？能说说每个手段的提升数据吗？
- 电商网站中，购物车的实时更新（如添加商品后数量同步）你是如何实现的？如何避免频繁请求导致的性能问题？

跨团队协作与业务理解

- 作为半个组长，你是如何分配任务的？如果团队成员对需求理解有分歧，你会怎么协调？
- 和设计师合作时，遇到过复杂动画或兼容问题吗？比如某款浏览器不支持渐变，你是怎么解决的？
- 在 Pageplug 项目中，为京东、顺丰定制组件时，客户需求不明确时你是怎么处理的？如何平衡技术实现和客户体验？

新技术与团队推动

- 你是如何快速学会 Shopify 应用开发的？有没有总结过学习方法？比如参考了哪些文档或社区资源？
- 你编写的新员工入职文档包含哪些内容？如何帮助新员工快速上手项目？有没有收到过反馈？
- 如果团队要引入一个新技术（如 Headless 架构），你会如何预研并推动落地？