项目介绍

🎯 「工具+应用」分层介绍与提问钩子

（开场白 - 清晰定义角色）

面试官您好，我过去的工作主要围绕一个**数字孪生平台展开**。我的工作可以清晰地分为两部分：

1. **平台工具开发**：开发平台底层的核心数据生产工具。
2. **平台应用开发**：基于我们平台的能力去构建上层的具体业务应用。

   技术栈以 **Vue 3 + vite** 和 **Element Plus** 为主，深度集成 **Mars3D** 和 **飞渡引擎**。

   在**平台工具**部分，我主要负责了三大模块，它们是整个平台中的重要功能：

3. **智能搜索模块**：它支持关键字搜索、取点后按范围画圆搜索、画多边形（或上传geojson文件）搜索三种搜索方式。搜索成功后，渲染支持分类的结果列表并使用mars3d或飞渡引擎的api在地图上进行打点。
4. **几何编辑器**：用户可以在Mars3D上绘制、编辑点、线、面、多点、多线、多面等地理要素，并进行图层树的管理。我**实现了一套撤销重做栈**，并最终将数据导出为GeoJSON格式。
5. **样式编辑器**：我**设计了一个支持“与或非”逻辑的规则引擎**，让地图能根据数据动态改变样式，用户无需编写复杂的 SLD 文件，就能通过可视化界面为GIS图层配置出强大的专题地图，打通了从配置到发布到 GeoServer 的完整流程。我认为这个是我独自完成的最复杂的一个项目。

   i. 首先我实现了一套**动态图层加载策略**，当图层数据量较小时使用 `GeoJSONLayer` 进行全量加载，当数据量较大时使用wfsLayer按需加载，通过**扩大请求的 BBOX 范围**来预加载周边数据，减少地图移动时的频繁请求和闪烁。在低地图层级（即视角高度很高）时，自动切换到 `WMSLayer` 来获取服务端渲染的图片，**提升全局浏览时的性能。**

   ii. 采用**面向对象编程**思想。**首先是最基础的样式配置类**，比如 `PointStyleEditor`、`LineStyleEditor`，它们只关心自己负责的几何类型的样式属性，并通过统一接口与上层通信。**核心是规则匹配类**，我设计了 `RuleEngine` 管理着规则的有序集合和执行策略。**然后是图层控制类**，它接收规则引擎的匹配结果，并调用 Mars3D 的底层API，将样式应用到地图上，控制图层的显隐和渲染。**最后是地图加载类**，它抽象了不同数据源（GeoJSON、WMS、WFS）的加载细节，为上层提供统一的数据接口。

   iii. 用户可以为同一图层创建多条样式规则，实现基于数据属性的专题地图。我设计了一个**树形结构** 来保存规则条件。支持字符串、数值、布尔值等数据类型，以及 `==`，，`<=` 等运算符，并可通过 **“与或非”** 逻辑运算符组合成复杂的过滤条件。规则按用户设定的顺序排序，**规则列表中后面的规则会覆盖之前的**，实现了类似 CSS 的“层叠”机制，因此系统为每个地图要素从下至上匹配规则。任何规则或样式的修改都会触发前端的**实时匹配计算和地图重绘。**

   在**平台应用**部分，我的工作是利用平台提供的能力来实现具体应用：

6. **场景编辑器**：这个应用允许用户在3D场景中添加模型、车流、植被、压平、挖洞等。我负责实现了**植被、车流等各种对象的属性编辑面板**，并全部集成到一个统一的编辑器中。这里我**设计了一套可扩展的面板结构**，来应对未来可能不断增多的对象类型。

7. **应用编排器**：这个更像一个**面向3D场景的PPT**，用户可以在底图上放按钮、图文，并设置点击按钮后触发相机漫游、显示/隐藏信息面板等。我实现了相关的可视化编排以及动画功能。

8. **智慧道路大屏**：这是作为**子应用嵌入在主平台**里的展示项目。它很好地体现了平台的能力，集成了**实时车流模拟**、各类数据统计面板（车流量、事件等），并实现了**自动路径漫游**功能，用于动态演示。

除此之外，我还参与了多个平台管理门户的开发：

- 使用公司内部前端框架，主要承担了各类**表格、表单**页面的开发，也处理过一些复杂的**树形结构**操作。这些工作虽然技术挑战性相对较低，但锻炼了我**快速完成业务需求**的能力，让我对后台系统的交互逻辑有了很深的理解。

🔍 新埋下的「钩子」与预期提问

1. **钩子1 (工具-深度)**： “**从头实现撤销重做栈**”

   - **预期问题**： “能聊聊你的撤销重做栈的设计吗？比如数据结构、如何应对复杂操作？”
   - **准备答案**：详细解释命令模式，栈如何管理 `{type, data, prevData}` ，以及如何处理边界情况（如内存控制）。
   - **回答：**
     - **数据结构：** 用一个数组（栈）来存储历史记录。每条记录包含： `{ type: 'add' | 'delete' | 'update', data: ..., previousData: ... }` 。
     - **执行过程：**
       - `add/delete` ： 执行时，将新增的Graphic对象或删除的对象快照存入栈中。撤销/重做时，反向执行删除/添加操作。
       - `update` ： 执行编辑时，保存元素的**前一状态**和**新状态**。撤销时，将元素恢复到 `previousData` ；重做时，再应用到新数据。
     - **性能优化：** 对连续的点移动等操作，我会进行**节流**或在其最终完成后才生成一条更新记录，避免栈被快速撑满。

2. **钩子2 (工具-深度)**： “**规则引擎性能提升优化**”

   i. **预期问题**： “如何进行优化的？”
   ii. **准备答案**：这是你的王牌，可以分层讲：从算法（规则决策树）、执行策略（防抖、增量更新）、到数据结构（空间索引）。

   iii. **回答：**
   “为了实现精准的增量更新，我设计了一套 **‘规则-要素反向索引’** 系统。核心是一个 `ruleFeatureIndexMap` 映射表，它记录了每条规则对应着哪些要素。

   - 当用户**只修改某条规则的样式**时（比如把颜色从红改成蓝），我 **不需要做任何规则匹配计算** 。我直接从这个映射表里查出所有匹配这条规则的要素，然后直接给它们换样式就行了，性能是O(1)定位 + O(n)更新，n只限于受影响的要素。
   - 当**规则条件发生变化**时，这个过程也很高效：

a. **清理**：我通过映射表快速找到所有‘老’要素，把它们标记为‘脏’。
b. **重匹配**：我通过空间索引等手段，快速找一个‘可能匹配’的要素子集，让它们去匹配新规则。
c. **更新**：最后更新映射表。

- **此外，在匹配算法上，我采用了从后往前的顺序，一旦匹配成功就‘短路’返回。** 这保证了匹配过程本身也是最优的。

3. **钩子3 (应用-广度)**：“**可扩展的面板架构**”

- **预期问题**：“你提到的可扩展面板架构是怎么设计的？”
- **准备答案**：阐述你如何用Vue的 `slot` 、动态组件或提供注册机制，来让新增一种对象类型的编辑面板变得简单。
- **回答：**
  - “它的核心是一个 **‘注册表模式’** 。我设计了一个中央注册表，任何新的编辑面板，比如‘模型面板’，只需要按照一个统一的interface实现自己的Vue组件，然后在系统启动时向这个注册表注册一下就可以。
  - 这个interface规定了panel的名称、对象类型，和对应的Vue组件
  - 然后，实现了一个**统一的万能容器**。这个容器负责：查看当前选中的对象类型，根据这个对象的类型，去中央注册表里查一下负责编辑的组件并动态渲染出来。