

**M1**



# Project Design: Sail (时空地图)

项目代号: Sail

项目愿景: 一个可视化的交互式历史百科, 用户通过滑动时间轴, 在地图上探索人类历史上的重大事件。

当前状态: MVP Step 2 完成 (前端交互与视觉打磨 / Frontend Polish Completed)

## 1. 技术栈选型 (Tech Stack)

组件	选型建议	理由
核心框架	<b>Next.js (App Router)</b>	React 框架, 处理页面路由和 API。客户端组件 (use client) 处理复杂交互。
数据库	<b>Firebase Firestore</b>	存储历史事件数据。目前仅作为只读数据源 ( <b>Read-Only</b> ), 暂不开放用户写入。
地图引擎	<b>Leaflet (react-leaflet)</b>	轻量级 2D 地图。支持自定义图层 (Panes) 和 DivIcon。
地图底图	<b>CartoDB Light</b>	极简风格, 突出数据内容。
状态管理	<b>React State / Refs</b>	处理高频动画(时间轴拖动)和复杂布局计算。
图标库	<b>Lucide React</b>	用于 UI 图标 (Zoom, Layers, Play controls)。

## 2. 核心数据模型 (Data Model)

为了解决历史数据的模糊性和地理区域的多样性, 我们采用了标准化的嵌套结构。

集合路径

artifacts/{\_\_app\_id}/public/data/events

### A. 时间结构 (ChronosTime)

解决“时间精度不一致”问题(如:某事件只知道年份, 某事件精确到日)。

```
interface ChronosTime {
```

```

year: number;      // 核心字段: 负数代表 BC, 0 映射为 1 AD (无 0 年)
month?: number;    // 1-12
day?: number;     // 1-31

// 精度标识: 决定 UI 如何格式化显示 (e.g., "1500 BC" vs "Jan 1, 2024")
precision: 'year' | 'month' | 'day' | 'hour' | 'minute';
}

```

## B. 地点结构 (ChronosLocation)

解决“地点精度”和“疆域范围”的表达问题。

```

interface ChronosLocation {
  lat: number;      // 中心锚点纬度
  lng: number;      // 中心锚点经度
  placeName?: string; // 自然语言地名 (e.g., "Roman Empire")

  // 空间尺度: 决定渲染时的默认半径或层级
  granularity: 'spot' | 'city' | 'territory' | 'continent';

  // 确信度: 决定边界样式 (Definite=实线, Approximate=虚线)
  certainty: 'definite' | 'approximate';

  // 可选: 自定义覆盖半径 (单位: 米)
  customRadius?: number;

  // 可选: 预定义区域 ID, 优先渲染复杂多边形 (e.g., 'roman_empire_117ad')
  regionId?: string;
}

```

## 3. 核心功能模块与交互逻辑 (详细规范)

### A. 地图模块 (The Map)

#### 1. 渲染层级 (Z-Index Strategy)

利用 Leaflet Panes 解决遮挡问题, 确保视觉层级正确:

- shapesPane (z=450): 区域轮廓(圆/多边形)。
- linesPane (z=550): 连接线(Leader Lines)和锚点(Dots)。
- cardsPane (z=700): 信息卡片。卡片永远在连线之上。

#### 2. 智能防重叠布局 (Smart Layout Algorithm)

- 聚类 (Clustering): 根据屏幕 X 坐标, 将距离过近的事件归为一组。

- **水平平铺 (Horizontal Spread)**: 计算簇的几何中心，强制将卡片向左右散开，确保无重叠。
- **垂直避让 (Vertical Stagger)**: 在密集区域，卡片垂直错落排列，防止遮挡邻近卡片的连线。
- 触发时机: 地图缩放结束 (zoomend)、平移结束 (moveend) 或时间变化时重新计算。

### 3. 连线系统 (Leader Lines)

- 视觉设计: 绘制从地图锚点 (lat, lng) 到卡片底部中心 的实线。
- 动态计算: 结合卡片内容高度(有图 vs 无图 )，精准计算连接点，避免线条“穿模”。

### 4. 空间过滤 (Spatial Filtering)

- 逻辑: 仅加载当前地图视口 (mapBounds) 内的事件。
- 交互: 拖动地图或缩放时，时间轴上的 Marker 会实时增减。如果已显示的卡片移出视野，它会自动消失。

### 5. 视觉表现

- 无级缩放: 启用 zoomSnap: 0, zoomDelta: 0.5, wheelPxPerZoomLevel: 10 以获得丝滑手感。
- 形状渲染:
  - regionId 存在 -> 渲染多边形 (目前使用前端硬编码字典，后续接 GeoJSON)。
  - 否则 -> 渲染圆形 (半径由 granularity 或 customRadius 决定)。

## B. 时间轴模块 (The Time Control)

### 1. 双层结构设计

- **主时间轴 (Detail View):**
  - 平滑动画: 点击空白处或 Marker，滑块平滑飞到目标点。
  - 交互优化: 拖动滑块时保持瞬时响应；点击跳转时启用动画。
  - **Visual Thumb**: 自定义滑块头 (z-40)，解决原生 Input 遮挡 Marker 问题。
- **概览时间轴 (Overview/Mini-map):**
  - 位于下方，固定显示全局历史范围。
  - **视口指示器 (Viewport Indicator)**: 可拖动的蓝色框，直观控制主时间轴的缩放和平移。

### 2. 缩放与导航 (Zoom & Pan)

- 缩放: 支持放大到 20 年 跨度，或缩小到 5000 年 跨度。
- 刻度自适应: 根据缩放级别，自动切换日期显示格式(年 -> 月 -> 日)。

### 3. 事件标记 (Event Markers)

- 在时间轴上显示小竖条/方块。
- 交互:
  - **Hover**: 瞬时高亮(无动画延迟)，显示小 Tooltip。
  - **Click**: 触发主滑块平滑跳转到该时间。

## C. 卡片交互 (Card Behavior)

- 触发模式: 被动显示 (**Passive**)。当滑块进入事件时间窗口时自动显示。
- 过渡效果:

- 出现 (Appear): 瞬时 (Instant)。0延迟, 确保快速滑动时也能被捕捉。
- 消失 (Disappear): 2秒淡出 (2s Fade-out)。提供视觉残留, 营造历史余韵。



## 4. 开发实施状态 (Project Status)

阶段	状态	详细进展
Step 1: 静态原型	✓ 完成	基础地图与滑块, Mock 数据展示。
Step 2: 前端体验	✓ 完成	<ol style="list-style-type: none"> <li>智能布局: 实现了卡片防重叠和连线避让。</li> <li>交互优化: 解决了滑块与 Marker 的层级冲突, 实现了平滑跳转。</li> <li>空间联动: 实现了基于地图视野过滤时间轴事件的功能。</li> <li>视觉系统: 完成了多边形区域高亮和卡片样式的标准化。</li> </ol>
Step 3: 数据接入	🚧 下一步	<p>目标: 将目前的 Mock Data 结构迁移至 Firebase Firestore。</p> <p>任务:</p> <ol style="list-style-type: none"> <li>在 Firestore 建立 events 集合。</li> <li>编写脚本批量导入当前的 Mock Data (包含 regionId 等新字段)。</li> <li>替换 MOCK_EVENTS 为 useFirestore 钩子读取数据。</li> </ol>

Step 4: 移动端适配	 待定	当前布局在窄屏手机上可能拥挤，需设计移动端专用视图。
---------------	--	----------------------------



## 接下来的步骤 (Next Steps)

1. **数据迁移 (Data Migration):**
  - 在 Firebase Console 中创建项目数据库。
  - 将代码中的 MOCK\_EVENTS 转换为 JSON，并编写脚本上传至 Firestore。
  - 注意：确保 ChronosTime 和 ChronosLocation 对象的结构在存储时保持一致。
2. **性能优化 (Performance):**
  - 当数据量从 50 增加到 500+ 时，当前的  $O(N^2)$  布局碰撞检测算法可能会卡顿。需要引入 QuadTree 或 Spatial Index 来优化碰撞检测。
3. **GeoJSON 服务化:**
  - 目前的 PREDEFINED\_REGIONS 字典存储在前端代码中，扩展性差。
  - 下一步应寻找轻量级的 GeoJSON API 或将边界数据存储在 Firebase Storage 中按需加载。

**M2**

# Sail Project (时空地图) - M2 里程碑技术归档

## 1. 里程碑概览

维度	状态	描述
阶段	M2 - 基础设施加固与核心功能实现	项目代码库已从原型升级为工程化产品，具备高并发抗性。
核心成就	代码结构分离、 <b>Supabase</b> 高效数据接入、复杂 LOD 过滤、 <b>Canvas</b> 动画和 <b>Unit Testing</b> 。	
下一阶段	M3 - 移动端适配及内容丰富化	

## 2. 架构重构与数据流 (Phase 1-3)

### 2.1 M2 最终文件结构 (Full Hierarchy)

```
project_root/
  └── app/
    ├── api/
    │   └── events/
    │       └── route.ts      # [Backend] Supabase RPC 调用, 含 NULL 短路修复
    └── page.tsx            # [Main] 主编排器, 负责 Hooks 组装和状态管理
  └── components/
    ├── debug/
    │   └── DebugHUD.tsx    # [UI] 实时监控面板 (M2 新增)
    ├── map/
    │   └── LeafletMap.tsx  # [UI] 地图渲染与事件绑定
    ├── panel/
    │   └── EventDetailPanel.tsx # [UI] 侧滑详情面板 (Master-Detail)
    └── timeline/
        ├── OverviewTimeline.tsx # [UI] Canvas 热力图渲染, 含高性能拖拽
        └── TimeControl.tsx     # [UI] 主时间轴控制器
  └── hooks/
```

```

├── useUrlSync.ts      # [Logic] URL 状态同步 (已修复无限循环 Bug)
├── useEventData.ts    # [Logic] SWR 数据获取, 含事件累积器与 Zod 校验
├── useLOD.ts          # [Logic] LOD 阈值计算 (Weighted Average)
└── useEventFilter.ts   # [Logic] 空间/重要性过滤管道 (Nearest Neighbor Projection)

lib/
├── _tests_/
│   ├── geo-engine.test.ts # [Test] 核心空间逻辑单元测试
│   └── time-engine.test.ts # [Test] 时间逻辑单元测试
├── constants.ts        # [Data] 预定义区域和 Mock 数据 (已弃用, 但保留结构)
├── geo-engine.ts        # [Math] 经度投影与空间计算的纯逻辑
├── schemas.ts          # [Validation] Zod 运行时校验 Schema
└── time-engine.ts       # [Math] BC/AD 时间数学计算

vitest.config.ts         # [Config] Vitest 配置

```

## 2.2 数据管线与核心逻辑 (Data Pipeline)

阶段	模块	职责与技术点
<b>DB &amp; Fetch</b>	route.ts & useEventData	高并发抗性。使用 Supabase RPC + CDN Cache (s-maxage=60)。后端统一处理全球/本地视野，前端使用 SWR 保持缓存。
<b>Validation</b>	useEventData + schemas.ts	数据安全。使用 Zod 在运行时校验 API 返回的数据结构。
<b>Filtering (LOD)</b>	useLOD + useEventFilter	去噪和性能。使用 加权平均策略 ( <b>Weighted Average</b> ) 同时依赖时间和缩放计算 Min Importance。
<b>Spatial Check</b>	useEventFilter + geo-engine.ts	地图健壮性。使用 最近邻投影 ( <b>Nearest Neighbor Projection</b> ) 算法，确保无限拖动地图时，事件点不会因坐标系包裹问题而丢失。
<b>UX/Performance</b>	OverviewTimeline.tsx	流畅性。拖拽使用 直接 DOM 操作 + rAF 节流，避免 React 重绘导致的卡顿。

### 3. M2 核心技术点与注意事项

#### 3.1 性能与交互 (Performance & UX)

问题描述	根本原因	优雅解决方案
拖拽滞后 (Drag Lag)	拖拽事件(MouseMove)以高频触发 React 状态更新，导致主线程阻塞。	rAF 锁定 + 直接 DOM 操作。视觉更新(指示器位置)直接通过 Ref 修改 DOM, 逻辑更新(setViewRange)被 requestAnimationFrame 锁定，确保更新频率不超过屏幕帧率(60fps)。
URL 刷屏	useEffect 依赖的 updateUrl 函数不断被重创建，导致路由陷入循环替换。	在 useUrlSync 中增加 1000ms 防抖，并使用字符串脏检查 (currentParams.toString() === newParams.toString())，只有 URL 真正变化时才调用 router.replace。
Marker 闪烁/消失	地图外的事件被 React 卸载 (Unmount)，无法播放 CSS opacity 动画。	事件累积器 (allLoadedEvents)。前端维护一个历史超集，确保 Marker DOM 节点始终存在，仅通过 isVisible 状态切换 CSS opacity: 0。

#### 3.2 数据库与地理空间 (PostGIS & Spatial)

问题描述	根本原因	优雅解决方案
全球视野数据缺失	1. 后端 API 曾尝试使用 select('*') 导致 WKB 几何数据解析失败。2. 全局边界 (-90 to 90 / -180 to 180) 导致 PostGIS 抛出 <b>Antipodal edge detected</b> 错误。	SQL NULL 短路。将 RPC 函数修改为：当处于全球或洲际视野时，前端传 min_lng = NULL。数据库通过 WHERE min_lng IS NULL OR (spatial check) 跳过空间过滤。彻底避免了使用

		-179.9 等不优雅的魔术数字。
无限拖动消失	前端简单数字比较 ( $10 \geq 350$ ) 无法识别经度的周期性。	最近邻投影 ( <b>Nearest Neighbor Projection</b> )。利用 $\text{Math.round}((\text{centerLng} - \text{eventLng}) / 360)$ 计算周期偏移量，将事件经度投射到当前屏幕视野内，保证无限世界地图上的事件永不丢失。
热力图偏差	热力图单纯基于线性权重，低重要性事件(数量多)在视觉上被高重要性事件(数量少)完全压制。	对数归一化 ( <b>Log Normalization</b> )。修正 Canvas 渲染算法，应用 Log 曲线增强低权重数据的可见度，实现视觉平衡。

### 3.3 调试与工具链

- 架构升级：逻辑代码 (useLOD, useEventFilter) 被彻底抽离为纯函数，为单元测试创造了条件。
- DebugHUD**：实时监控 LOD 阈值、Fetched/Rendered 计数和 GLOBAL/LOCAL 模式，将复杂的多维过滤问题转化为直观的视觉检查。
- Zod** 集成：在数据获取的边界处进行运行时类型校验，保障前端应用对来自 API 的脏数据的鲁棒性。

**M3**

# Sail Project (时空地图) - M3 阶段技术演进报告

日期: 2025-12-03

状态: M3 进行中 (Data Pipeline & Scalability)

重点: 自动化数据流水线、双环境架构、高性能可视化、复杂时空查询

## 1. 核心架构演进

### 1.1 混合智能流水线 (Hybrid Intelligence Pipeline)

为了支撑从 10万+ 到百万级的历史数据, 我们确立了 **Python Backend (ETL) + Next.js Frontend (Display)** 的双栈架构。

- 设计哲学: “本地离线处理 (Backfill)” 与 “线上增量更新 (Incremental)” 分离, 但在 data-pipeline/src/lib 中共享核心清洗逻辑。
- ETL 流程:
  - Extract:** 从 DBpedia/Wikidata 获取原始数据。
  - Transform:**
    - 时间归一化: 将 ISO 格式转换为自定义的 astro\_year (Decimal Year)。
    - 地理归一化: 将 WKT 转换为 PostGIS 格式。
    - 评分 (**Scoring**): 基于规则(词条长度、关键词)计算初始 importance。
  - Load:** 使用 upsert (基于 source\_id 去重) 写入 Supabase。

### 1.2 多环境隔离 (Environment Isolation)

为了保证生产环境数据的稳定性, 我们实施了严格的数据隔离策略。

- 数据库层:
  - events\_prod: 生产表, 仅存储清洗验证后的数据。
  - events\_dev: 开发表, 用于测试爬虫和算法。
  - 双 **RPC** 策略: 部署了 get\_events\_in\_view\_v2 (查 Prod) 和 get\_events\_in\_view\_dev (查 Dev)。
- 应用层:
  - 配置中心 (**useAppConfig**): 移除 URL 参数控制, 严格通过环境变量 (NEXT\_PUBLIC\_DATASET) 或构建模式 (NODE\_ENV) 决定读取哪个数据源。
  - 启动脚本: npm run dev (读 Dev) vs npm run dev:prod (读 Prod)。

## 2. 高级数据模型 (Advanced Data Model)

我们重新定义了时间和空间的数据结构, 以适应历史的复杂性。

### 2.1 双轨时间系统 (Dual-Track Time)

- Wall Time (显示层):** year, month, day, second, millisecond。永远记录事件发生地的当地时

间，不进行 UTC 转换，保留历史语境。

- **Astronomical Year** (计算层): astro\_year。
  - 定义: 连续的线性小数年份。
  - 规则: 1 AD = 1.0, 1 BC = 0.0, 2 BC = -1.0。
  - 用途: 数据库索引、排序、热力图密度计算。

## 2.2 幂等性与来源

- **source\_id**: 引入自然键 (Natural Key)，例如 dbpedia:Battle\_of\_Waterloo。这是数据流水线实现 **Upsert** (幂等写入) 的基础。

# 3. 核心功能实现与优化

## 3.1 后端 API 与 PostGIS 优化 (route.ts)

- 全球视野短路 (**Global View Short-Circuit**)：
  - 问题: PostGIS 处理跨越 180 度经线的查询时会抛出 Antipodal edge detected 错误。
  - 解决: 智能判断 Zoom < 5.5 或经度跨度 > 160 度时，向数据库传 NULL 参数。SQL 函数内部检测到 NULL 则跳过 ST\_MakeEnvelope 计算，直接返回全量数据。
- 数据清洗: 后端 API 负责将数据库的 snake\_case 字段映射为前端的 camelCase 字段，并处理 start\_time\_body JSON 的回退逻辑。

## 3.2 高性能概览热力图 (OverviewTimeline.tsx)

- **Canvas** 渲染: 放弃 CSS，使用 HTML5 Canvas 实现每秒 60 帧的渲染。
- 视觉算法：
  - 高斯平滑 (**Gaussian Smoothing**): 使用宽核 (Radius=12, Sigma=5) 将离散事件点晕染为连续的密度波峰。
  - 对数归一化 (**Log Normalization**): 使用  $\log(\text{val})/\log(\max)$  提升低频事件的可见度，防止大事件“独占”热力图。
  - 液态动画: 使用 Lerp 插值，在数据变化时实现热力图形状的平滑流变。
  - 性能: 拖拽指示器时采用直接 DOM 操作 (0 延迟) + rAF 锁定 (逻辑节流) 的双层更新机制。

## 3.3 健壮的前端空间过滤 (page.tsx)

- 无限地图投影 (**Nearest Neighbor Projection**): 解决了 Leaflet 允许无限拖动导致坐标超出 [-180, 180] 的问题。算法自动将事件经度投影到离当前视口最近的“平行世界”中。
- **LOD (Level of Detail)**: 采用 加权平均策略  $((\text{TimeLOD} + \text{MapLOD}) / 2)$  动态计算重要性阈值，在“性能卡顿”和“地图空旷”之间取得了最佳平衡。

# 4. 当前项目文件结构 (M3 Snapshot)

```
sail-project/
  └── app/
    └── api/
      └── events/
```

```
|   └── route.ts      # [Backend] 核心 API, 含环境路由与 PostGIS 优化
|   └── page.tsx       # [Main] 前端编排器 (LOD, Filter, SWR)
|   └── components/
|       └── debug/
|           └── DebugHUD.tsx    # [UI] 调试面板
|       └── map/
|           └── LeafletMap.tsx  # [UI] 地图组件
|       └── timeline/
|           └── OverviewTimeline.tsx # [UI] Canvas 高斯热力图 + 高性能拖拽
|               └── TimeControl.tsx  # [UI] 主时间轴
|       └── panel/ ...
|   └── data-pipeline/    # [M3 Core] Python ETL 系统
|       └── .env          # 独立环境变量
|   └── requirements.txt
|   └── src/
|       └── backfill/     # 离线任务
|           ├── fetch_raw.py # 数据抓取
|           └── process_data.py # 数据清洗
|       └── lib/          # 共享核心逻辑
|           ├── time_normalizer.py
|           └── geo_normalizer.py
|   └── tests/          # Python 单元测试
|   └── hooks/
|       └── useAppConfig.ts      # [Config] 环境配置 (Dev/Prod)
|       └── useEventData.ts     # [Data] SWR + Zod 校验
|       └── useEventFilter.ts   # [Logic] 投影算法
|       └── useLOD.ts          # [Logic] 阈值计算
|       └── useUrlSync.ts      # [Logic] URL 同步
|   └── lib/
|       └── schemas.ts        # [Validation] Zod Schemas
|   └── ...
|   └── types/
|       └── index.ts         # [Core] 类型定义 (含 astro_year)
```

## 5. 待办事项 (Next Steps)

1. 大规模数据导入: 运行 2\_process\_data.py 和 3\_upload.py (待编写), 将抓取的 7000+ 条数据灌入 events\_dev。
2. LLM 集成: 在 ETL 流程中接入 LLM, 对 summary 进行润色, 并校准 importance 评分。
3. 前端性能优化: 面对数万条数据, 评估是否引入 Supercluster 进行地图聚合。