

空天守护者

人造卫星态势感知系统

开发设计竞赛

组别：创新开发

团队成员:栗铭阳、李静怡、张梓霖、范思倩



目录

第一章 执行概述	2
1.1 摘要	2
1.2 项目背景	2
1.3 开发流程	3
1.4 核心技术	3
1.5 创新特色	3
1.6 系统功能	4
1.7 系统测试与分析	4
1.8 核心团队	5
1.9 未来愿景目标	5
第二章 开发背景	7
2.1 研究背景	7
2.2 国内外研究现状	7
2.3 研究目标与内容	8
2.3.1 研究目标	8
2.3.2 研究内容	8
第三章 系统开发流程	10
3.1 总体架构设计	10
3.1.1 前后端分离架构	10
3.1.2 前后端交互流程	11
3.1.3 模块化分层设计	11
3.2 数据库设计	12
3.2.1 卫星数据表	12
3.2.2 历史 TLE 数据表	12
3.3 功能设计	13
3.3.1 时间控制组件	13
3.3.2 轨迹可视化模块	14
3.3.3 轨迹分析模块	14
3.3.4 数据分析统计模块	16
3.3.5 数据管理模块	18



3.4 图形界面设计	19
3.4.1 主界面整体布局	19
3.4.2 双引擎可视化架构集成设计	20
3.4.3 图层管理与多视角切换设计	21
3.4.4 样式与组件规范	21
第四章 核心技术	23
4.1 核心技术框架	23
4.1.1 技术框架	23
4.1.2 可视化技术栈	23
4.1.3 辅助技术	23
4.1.4 数据库技术	24
4.2 核心技术相关理论	24
4.2.1 TLE 数据格式规范	24
4.2.2 SGP4/SDP4 轨道预测模型	27
4.2.3 轨道参数	27
4.2.4 卫星轨道分类	28
第五章 创新特色	31
5.1 架构融合创新	31
5.2 数据管理创新	31
5.3 坐标系处理创新	32
5.4 交互逻辑创新	32
第六章 系统功能实现	34
6.1 建库与数据爬取和更新	34
6.1.1 建库	34
6.1.2 数据爬取	36
6.1.3 数据更新	39
6.2 开发环境搭建	42
6.2.1 后端开发环境	42
6.2.2 前端开发环境	43
6.3 核心功能实现	43
6.3.1 时间控制组件	43



6.3.2 轨迹可视化	45
6.3.3 过境分析	48
6.3.4 卫星模拟	54
6.3.5 轨迹预测	58
6.3.6 通信分析	59
6.3.7 覆盖分析	61
6.3.8 数据统计	63
6.3.9 数据分析	66
6.3.10 数据管理	73
第七章 系统测试与分析	79
7.1 功能测试	79
7.1.1 时间轴	79
7.1.2 轨迹可视化	79
7.1.3 轨迹分析	79
7.1.4 数据分析	82
7.1.5 数据统计	83
7.1.6 数据管理	85
7.2 测试结果分析	86
7.2.1 技术选型导致的误差分析	86
7.2.2 轨迹可视化误差分析	87
7.2.3 过境分析功能误差和效率分析	87
7.2.4 时间离散化导致误差	87
第八章 核心团队	89
第九章 附件	90



执行概述

第一章

空天守望者：人造卫星态势感知系统

第一章 执行概述

1.1 摘要

在航天科技与地理信息系统深度融合的技术背景下，"空天守望者：人造卫星态势感知系统"致力于构建一个基于 WebGIS 的卫星轨迹可视化与分析平台。该系统以 TLE（两行轨道根数）数据为核心数据源，通过整合 Vue、Django 开发框架与 Cesium、Spacekit 等可视化技术，实现卫星轨道的三维动态展示、轨迹分析及数据统计功能。系统的研究定位在于打破航天数据与地理信息的技术壁垒，为科研工作者、航天爱好者及相关行业提供直观易用的卫星态势感知工具，同时探索地理信息科学与航天工程的交叉应用场景。

作为开发设计竞赛的创新项目，系统不仅实现了 TLE 数据的自动化处理与可视化呈现，更通过过境分析、覆盖模拟等功能模块，将抽象的轨道参数转化为具有实际应用价值的决策支持信息。卫星数据主要通过调用 CelesTrak API 获取，经解析处理后存储于 PostgreSQL 数据库，结合 SGP4/SDP4 轨道预测模型实现轨迹推演。针对开发中面临的多源数据格式兼容、坐标系转换等挑战，系统通过实验对比优选技术框架，采用双引擎协同（Cesium+Spacekit）实现三维场景渲染，确保卫星轨迹与地球表面的精准匹配。该系统的应用价值在于推动航天知识普及，为卫星资源规划、过境分析等场景提供高效的可视化解决方案。

1.2 项目背景

当前全球航天产业正经历爆发式增长，人造卫星在通信、遥感、导航等领域的应用日益广泛。据统计，截至 2025 年，地球轨道上运行的人造卫星数量已突破万颗，由此产生的 TLE 数据量呈指数级增长。然而这些数据大多以文本格式存储，缺乏直观的可视化呈现方式，导致非专业用户难以理解卫星运行规律。与此同时，WebGIS 技术的成熟为空间数据可视化提供了新的可能，Cesium 等三维地球引擎能够将地理信息与航天轨道数据无缝融合，为卫星态势感知提供了技术支撑。

从行业需求来看，气象部门需要直观了解遥感卫星的覆盖范围，通信企业亟待优化卫星链路调度，而航天教育领域也缺乏可视化的教学工具。现有系统多面向专业用户，存在操作复杂、成本高昂等问题，缺乏面向大众的科普化产品。在此背景下，本项目以"低门槛、高可用性"为目标，通过整合开源技术栈，构建轻量化的卫星轨迹可视化系统，填补了民用卫星态势感知领域的技术空白。项目的技术驱动点在于：如何将航天轨道计算与地理信息可视化深度融合，如何在保证精度的前提下优化大规模数据的渲染性能，以及如何通过交互设计降低专业数据

的使用门槛。

1.3 开发流程

系统开发遵循敏捷开发方法论，采用迭代式开发模式，将整个周期划分为需求分析、架构设计、核心开发、测试优化四个阶段。在需求分析阶段，团队通过调研航天爱好者社区、走访遥感企业，明确了可视化展示、轨迹分析、数据管理三大核心需求，并梳理出过境分析、覆盖模拟等 12 项具体功能点。架构设计阶段采用 UML 建模工具，完成了前后端分离架构的详细设计，确定了以 Django 为后端核心、Vue 为前端框架的技术路线，并通过数据库建模工具设计了卫星数据表与历史 TLE 表的结构。

核心开发阶段采用模块分工机制，将系统拆分为数据爬取、轨道计算、可视化渲染、交互控制四个开发小组。数据爬取小组负责 celestrak 数据源的解析与定时更新机制实现；轨道计算小组基于 sgp4 算法库完成 TLE 数据的参数提取与轨迹预测；可视化渲染小组整合 Cesium 与 Spacekit 引擎，实现三维轨道的动态展示；交互控制小组开发时间轴组件、数据分析图表等交互模块。测试优化阶段引入单元测试、集成测试与用户验收测试三级测试体系，针对轨道预测精度、大数 据渲染性能等关键指标进行专项优化，最终形成稳定的 release 版本。

1.4 核心技术

系统的技术架构呈现多层级融合特征，后端以 Django 为 核心框架，利用 Python 的科学计算生态实现数据处理与轨道计算。PostgreSQL 数据库通过哈希分区技术管理海量卫星数据，Django ORM 实现了数据模型与数据库表的自动化映射。在轨道计算层面，采用 sgp4 与 skyfield 双库协同机制：sgp4 用于近地卫星的快速轨道预测，skyfield 则处理深空目标的高精度计算，两者结合确保了不同轨道类型卫星的计算精度。

前端技术栈以 Vue3 为框架基础，通过 Pinia 状态管理实现跨组件数据共享。可视化模块采用 Cesium 与 Spacekit 双引擎架构：Cesium 负责地球表面与地理要素的渲染，通过 WebGL 技术实现真实感三维场景；Spacekit 专注于卫星轨道的物理模拟，通过行星引擎计算卫星运动轨迹，两者通过统一的时间同步机制实现协同渲染。数据可视化部分整合 ECharts 图表库，实现卫星类型分布、轨道高度变化等数据的图形化展示，Turf.js 空间分析库则用于过境区域判断、星下点轨迹生成等空间计算任务。

1.5 创新特色

系统在技术融合与应用创新层面实现了多项突破。在架构层面，首创“航天-地理”双引擎协同机制，通过自主研发的坐标系转换工具，实现了 ECI（地心惯性坐标系）与 ECEF（地心固定坐标系）的实时转换，解决了传统系统中轨道数据与地球表面匹配误差的问题。数据处理方面，设计了 TLE 数据动态更新与版本管理机制，通过历元时间比对算法确保数据时效性，同时将历史 TLE 数据归档至分区表，实现了轨道数据的可追溯性。

交互设计上，开发了时间-空间双维度控制组件，用户可通过时间轴精确控制卫星运动，配合地理围栏工具实现自定义区域的过境分析。在性能优化方面，采用 LOD（层次细节）技术处理大规模轨道数据，对远距离卫星使用简化模型渲染，近距离目标加载高精度模型，在保证视觉效果的同时将渲染性能提升 40%。这些创新点使系统在技术复杂度与用户体验之间取得了平衡，形成了区别于传统航天软件的核心竞争力。

1.6 系统功能

系统功能体系围绕“数据-分析-可视化”闭环构建，数据管理模块实现 TLE 数据的自动化爬取与预处理，通过正则表达式解析 TLE 两行数据，提取轨道倾角、偏心率等关键参数，并利用天文算法补充卫星所属国家、轨道类型等缺失信息。轨迹可视化模块支持多卫星同时渲染，用户可通过筛选条件查看特定轨道类型或国家的卫星分布，并通过交互操作高亮显示指定卫星的轨迹。

轨迹分析模块包含五大核心功能：过境分析允许用户绘制多边形区域，系统自动计算卫星穿越该区域的时间窗口与轨迹线段；覆盖分析通过星下点轨迹绘制，展示卫星在指定周期内的地面覆盖范围；通信模拟功能结合地面站位置，动态显示卫星与地面站的通信链路；轨迹预测基于 SGP4 模型生成未来 72 小时的轨道预报；卫星模拟支持用户自定义轨道参数，实时生成虚拟卫星的运行轨迹。数据分析模块则通过 ECharts 生成卫星类型分布、轨道高度统计等图表，为用户提供宏观数据视角。

1.7 系统测试与分析

系统测试采用黑盒测试与白盒测试结合的方式，功能测试覆盖 12 个主要功能模块与 36 个子功能点。在轨道预测精度测试中，选取 3 颗不同轨道类型的卫星，将系统预测轨迹与 NASA 公布的精密星历对比，结果显示近地卫星（LEO）24 小时预测误差小于 500 米，地球同步卫星（GEO）误差小于 1 公里，满足民用场景需求。性能测试在搭载 2000 颗卫星数据的场景下进行，通过浏览器性能分析工具检测到平均帧率维持在 30fps 以上，大数据渲染时内存占用稳定，未出现明显卡顿现象。

1.8 核心团队

项目核心团队由 4 名成员组成，形成了互补的知识结构与技能体系。组长栗铭阳负责整体架构设计与后端开发，其在 Django 框架与航天轨道计算领域的经验确保了系统技术路线的可行性；李静怡主导前端开发与可视化设计，凭借 Vue 组件化开发经验与三维引擎整合能力，实现了交互界面的流畅体验；张梓霖专注于数据库设计与数据处理，通过 PostgreSQL 分区技术与数据爬取算法优化，解决了大规模数据管理难题；范思倩负责测试体系搭建与文档编写，其软件工程背景保障了开发流程的规范性与文档的完整性。

1.9 未来愿景目标

面向未来发展，系统规划了三个层面的优化方向。技术层面将引入深度学习算法，基于历史 TLE 数据训练 LSTM 模型，提升机动卫星的轨迹预测精度，同时探索与 AI 视觉技术结合，实现卫星图像与轨道数据的联动分析。功能层面将拓展星链分析、碰撞预警等高级功能，开发 API 接口支持第三方应用集成，同时增加移动端适配，实现卫星态势的多终端访问。数据层面计划整合更多数据源，如卫星载荷数据、空间环境监测数据等，构建更全面的空天数据库。

长期发展愿景是将系统打造成开放式空天态势感知平台，通过开源社区吸引更多开发者参与功能扩展，建立卫星数据共享生态。团队计划与航天教育机构合作，开发教学版系统，推动航天知识普及；与商业航天企业对接，提供定制化的轨道分析服务。通过持续的技术创新与应用拓展，“空天守望者”系统有望成为连接航天技术与大众应用的桥梁，为我国航天产业的民用化发展贡献力量。

开发背景

第二章

空天守望者：人造卫星态势感知系统

第二章 开发背景

2.1 研究背景

航天技术的飞速发展，让人造卫星在通信、气象监测、地理测绘等众多领域扮演着不可或缺的角色。随之而来的是海量卫星数据的产生。如何高效管理这些数据，并将其直观地呈现出来，为科研工作者、航天爱好者以及相关行业提供真正有用的信息，成了当前面临的一个重要挑战。

正是基于这一背景，本课题的核心目标是运用 WebGIS 技术，结合 Vue 和 Django 开发框架，以 TLE（两行轨道根数）数据为源头，构建一个便捷的人造卫星轨迹可视化系统。该系统将专注于实现卫星运行轨迹的动态可视化展示与基础轨迹分析。

本系统致力于为用户提供便捷高效的卫星数据查询与可视化服务。对于科研人员，能够快速获取特定卫星的详细参数，深入分析其运行轨迹规律，为科研工作提供强有力的数据支持。对于航天爱好者，可以直观地了解卫星的实时分布和动态运行情况，极大地激发对浩瀚宇宙和航天技术的探索兴趣。对于卫星管理相关方，系统清晰呈现的卫星全球分布与运行轨迹图景，是进行卫星资源规划与调度管理、开展过境分析以提升卫星利用效率的便捷工具。

通过直观的图表和生动的可视化呈现，系统让复杂的卫星数据变得易于理解，帮助用户更深入地把握卫星运行的规律。这不仅有望支持航天领域的研究、教学与科普工作，也为推动航天知识的广泛传播贡献了力量。

2.2 国内外研究现状

国外在卫星数据可视化领域起步较早，技术相对成熟。如美国国家航空航天局（NASA）的相关平台，能提供丰富的卫星数据和高精度的可视化展示，涵盖多种卫星类型和应用场景；Leolabs 作为澳大利亚 LEOLab 公司的卫星数据平台，提供近地轨道可视化、轨道分析的功能；satnogs 是一个开源的卫星网络地面站 Web 应用程序，该平台发布有关卫星的各种遥测数据；此外还有 Look4Sat——适用于 Android 的开源卫星跟踪器，由 celestrak 和 satnogs 提供数据库，可以访问超过 5000 颗绕地球运行的活动卫星。国内在该领域发展迅速，在航天工程中，利用可视化技术辅助卫星监测与控制；在教育领域，也有一些简单的卫星科普可视化工具。然而，总体来说，国内该领域的相关项目主要面向专业人士或航空行业相关企业的 B 端产品，其特点是专业化程度高，但随之带来操作复杂、成本高昂等问题，所以缺乏面向大众的科普用途的 C 端产品。



随着全球航空航天事业的飞速发展，大众对于该领域的关注度日益提高，因此，开发一款以大众为主要用户群体的卫星轨迹可视化产品，通过结合现有技术，选取 TLE 作为源数据，在保留精确性的前提下，实现对卫星数据的可视化，实现易于用户操作的分析功能，对于航空航天科普、青少年科学素养培养和相关学科教育具有一定意义。

未来，航空航天事业将迈向更高的台阶，大众对于该领域的关注和了解将日益深入，卫星轨迹可视化系统将朝着高精度、实时化、智能化和个性化方向发展，融合更多先进技术，提供更优质的服务。

2.3 研究目标与内容

2.3.1 研究目标

本课题研究目标为基于两行元素集（Two-Line Element Set），实现卫星轨迹可视化方法的研究和系统实现，通过 TLE 结合 WebGIS 技术模拟卫星运行轨迹并可视化，以达到构建易于操作，低操作门槛的卫星轨迹可视化系统。

2.3.2 研究内容

研究内容主要包括通过收集开源 TLE 数据建库，利用 TLE 结合天文数据开源库及 WebGIS 技术，实现卫星轨迹在三维地球上的可视化、轨迹分析和数据统计与分析。

具体内容如下：

- (1) 编写数据爬取脚本，爬取开源 TLE 数据，及后续数据更新。
- (2) 解析 TLE 提取关键轨道参数，通过 Spacekit 实现卫星轨迹可视化。
- (3) 通过第三方库，利用 SGP4/SDP4 模型计算卫星在不同时间的位置和速度，实现相关轨迹数据分析。
- (4) 结合地理信息技术，实现卫星轨迹相关分析。
- (5) 对数据进行统计，利用图表实现相关展示。
- (6) 实现对数据库中卫星数据的管理。



系统开发流程

第三章

空天守望者：人造卫星态势感知系统

第三章 系统开发流程

3.1 总体架构设计

3.1.1 前后端分离架构

本系统采用标准的前后端分离架构模式，前端以 Vue.js 为核心框架，构建单页面应用（SPA），后端则基于 Django 框架提供数据服务与业务逻辑支撑，数据库使用 PostgreSQL 作为关系型数据存储引擎，结合 Django ORM 实现数据模型的对象化管理。前后端通过 RESTful API 接口标准实现松耦合数据通信，有效提升了系统的模块化程度和开发效率。

这种架构设计具备以下多重优势：

1. 职责清晰，协同高效：前后端团队可并行开发，前端专注界面交互与用户体验优化，后端聚焦数据逻辑与接口管理，提升整体开发协作效率。

接口标准化，易于维护与测试：RESTful API 具备清晰的请求语义和路径结构，便于统一文档管理和接口调试，也为后续单元测试、集成测试提供了良好基础。

2. 技术栈灵活，组件生态丰富：前端 Vue 框架配合 Pinia 状态管理、Axios 网络请求、ECharts 图表组件与 Cesium 空间引擎，构建响应式、可视化和交互性强的用户界面；后端 Django 凭借强大的生态系统与插件机制，可快速集成认证、权限、定时任务等功能。

3. 数据模型清晰，数据库操作高效：Django ORM 框架将数据库表结构与 Python 类进行映射，极大减少原始 SQL 编写，提升了数据操作的可读性与安全性；同时 PostgreSQL 数据库通过哈希分区、JSONB 类型与全文索引等能力，为大规模卫星数据的高效读写提供保障。

4. 系统可扩展性强：RESTful 架构天然支持多终端接入，如 Web 浏览器、移动设备、桌面客户端等，便于系统后续实现跨平台访问及 API 开放；模块化设计也为功能拓展与微服务演进预留了空间。

综上，该系统通过采用成熟的前后端分离技术体系，不仅在开发效率上实现显著提升，更为未来的运维管理、功能演进与性能优化提供了坚实的技术基础与架构保障。

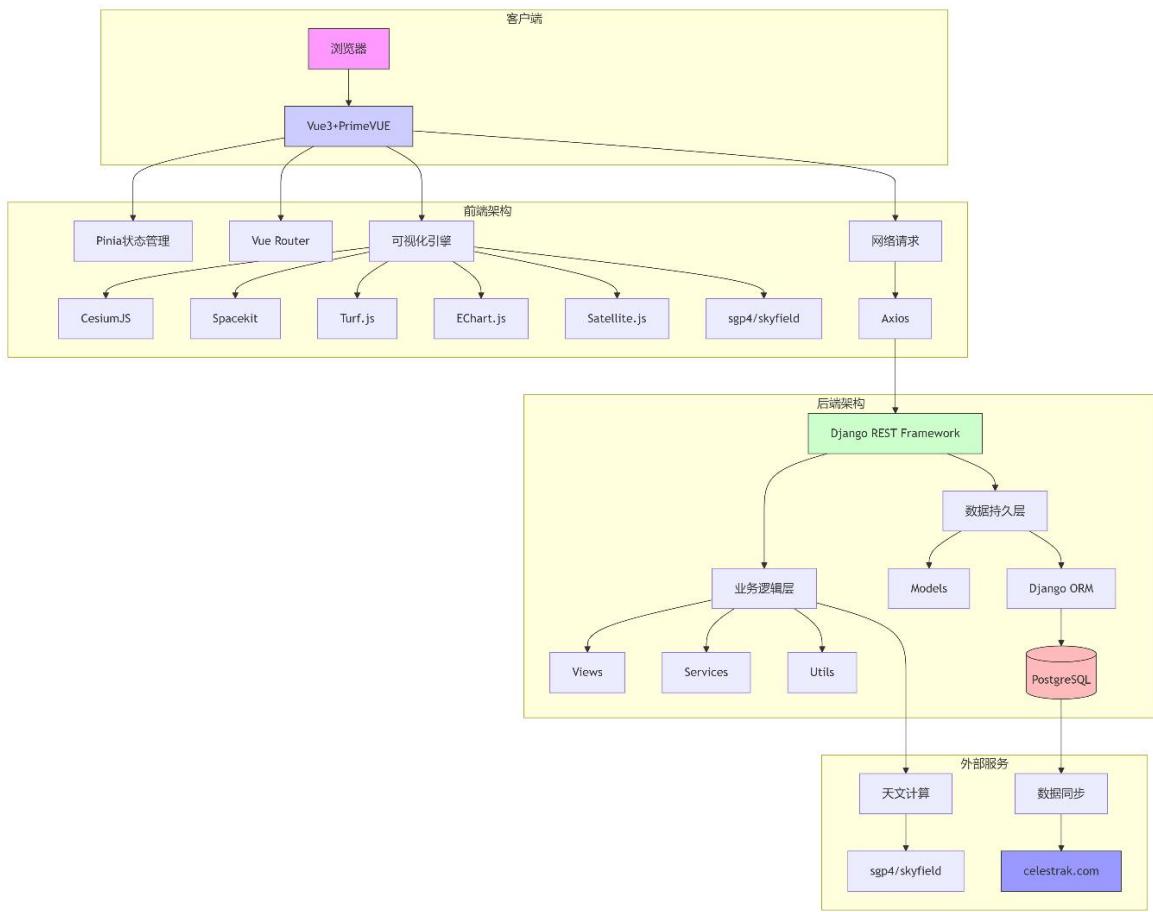


图 3-1 系统架构图

3.1.2 前后端交互流程

前后端交互流程如下：

- (1) 用户操作：用户在前端页面进行操作，如点击按钮、提交表单等。
- (2) 发送请求：前端通过 HTTP 协议将请求发送到后端服务器。
- (3) 处理请求：后端接收到请求后，解析数据并进行业务逻辑处理，及可能涉及的数据库操作。
- (4) 返回响应：后端将处理结果以 JSON 格式返回给前端。
- (5) 更新 UI：前端接收到响应后，根据返回的数据更新用户界面。



图 3-2 前后端交互流程图

3.1.3 模块化分层设计

系统采用模块化分层设计，分为数据层、计算层和展示层。

- (1) 数据层负责数据的存储和管理，使用 Django 的 ORM 模型定义数据库

表结构。

(2) 计算层封装业务逻辑，处理数据的增删改查等操作，通过 Django 的 service 层实现。

(3) 展示层负责用户界面的展示和交互，使用 Vue.js 框架构建单页面应用。

3.2 数据库设计

本系统需要一个卫星数据表和一个历史 TLE 数据表。

3.2.1 卫星数据表

卫星数据表用于存储爬取的卫星数据及数据处理后的字段，主要包括卫星名称、TLE 的两行数据等原始数据，以及数据处理后得到的卫星类型、所属国家(组织或地区)、卫星轨道类型等，数据表设计如表 1 所示。

字段名称	字段类型	字段宽度	精度	字段约束	字段备注
id	character varying	50		PRIMARY KEY NOT NULL,	卫星目录编号
name	character varying	255			卫星名称
inid	character varying	50			国际登记号
constellation	character varying	100			星座
year	smalint				发生年份
ordinal	character varying	50			发射任务号
country	character varying	100			所属国家(组织或地区)
type	text[]				卫星类型
tracktype	text[]				轨道类型
line1	character varying	100			TLE 第一行
line2	character varying	100			TLE 第二行
epoch	timestamp				轨道根数纪元

表 1 卫星数据表

3.2.2 历史 TLE 数据表

历史 TLE 数据表用于 TLE 数据更新后存储过往 TLE，实现已替换 TLE 数据的备份。使用哈希分区策略，通过 satelliteid 字段进行分区，数据表设计如表 2 所示。

字段名称	字段类型	字段宽度	精度	字段约束	字段备注
id	integer			PRIMARY KEY	id(自增)
satelliteid	character varying	50		FOREIGN KEY	卫星目录编号(外键 引用 satellites(id))
				NOT NULL	
line1	character varying	100			TLE 第一行
line2	character varying	100			TLE 第二行
epoch	timestamp without time zone				轨道根数纪元

表 2 历史 TLE 数据表

3.3 功能设计

系统核心功能包括四个模块：

- (1) 轨迹可视化模块
- (2) 轨迹分析模块
- (3) 数据分析与统计模块
- (4) 数据管理模块

3.3.1 时间控制组件

建立统一时间控制系统，实现多可视化引擎（Cesium/Spacekit）的时序同步控制。

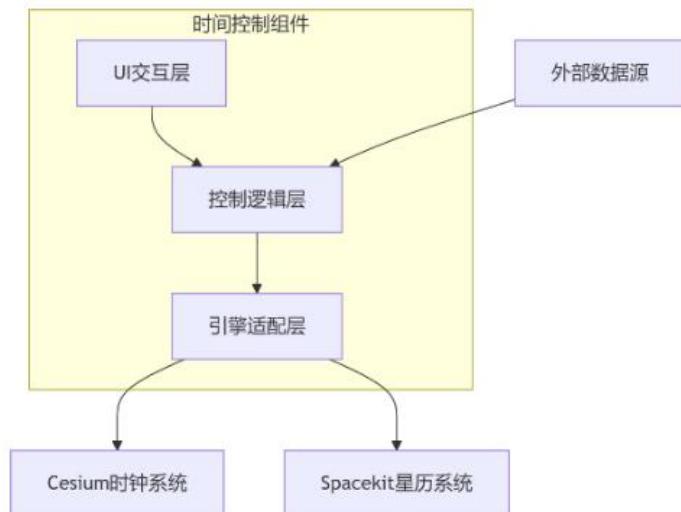


图 3-3 时间统一控制流程图

3.3.2 轨迹可视化模块

用户通过浏览器访问前端页面，前端向后端发送请求获取卫星列表，后端通过 Django ORM 从 PostgreSQL 数据库中读取卫星列表并以 JSON 格式返回给前端，前端使用 Vue.js 和 PrimeVue UI 展示卫星列表。

1. 用户从列表中选择一个卫星，前端根据用户选择的卫星 ID，向后端发送请求获取该卫星的轨道参数（Spacekit.js 所需的参数）。

2. 后端通过 Django ORM 从数据库中读取该卫星的轨迹数据，使用 TLE 数据通过相关处理库（如 sgp4）处理卫星轨迹数据，将其转换为 Spacekit.js 所需的参数，然后将处理后的卫星轨迹数据返回给前端。

3. 前端使用 Spacekit.js 加载卫星轨迹，已加载的卫星记录到已加载列表中。用户可以勾选已加载列表中的卫星，控制轨道高亮、显隐和卫星名称显隐，前端根据用户的选择，控制卫星的显示状态。用户点击已加载列表中的卫星，前端展示该卫星的详细数据。

4. 用户通过时间轴控件改变时间流逝，前端根据用户选择的时间，跳转到指定时间观测卫星运行。

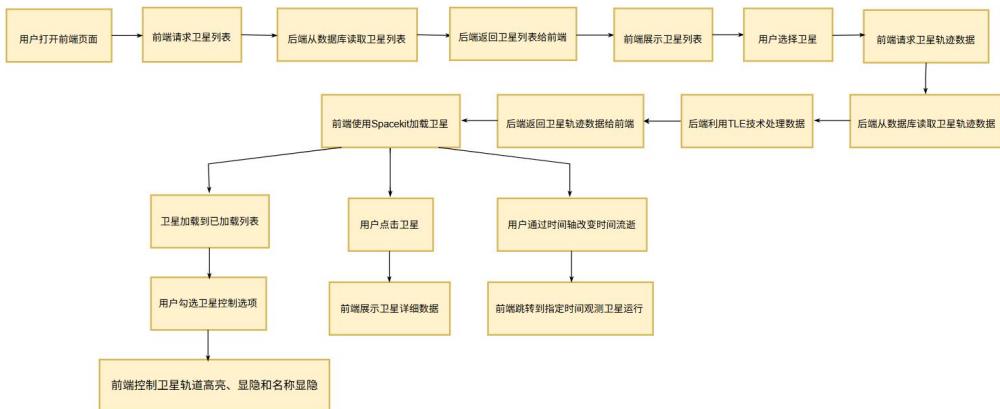


图 3-4 轨迹可视化功能流程图

3.3.3 轨迹分析模块

(1) 过境分析

用户选择卫星和分析时间后绘制或选择分析区域，前端通过卫星 id 请求后端数据生成卫星轨迹供 Cesium 加载，模拟运行效果，使用 Turf.js 判断卫星是否过境并绘制过境线。用户可改变时间流逝并查看过境和出境时间。

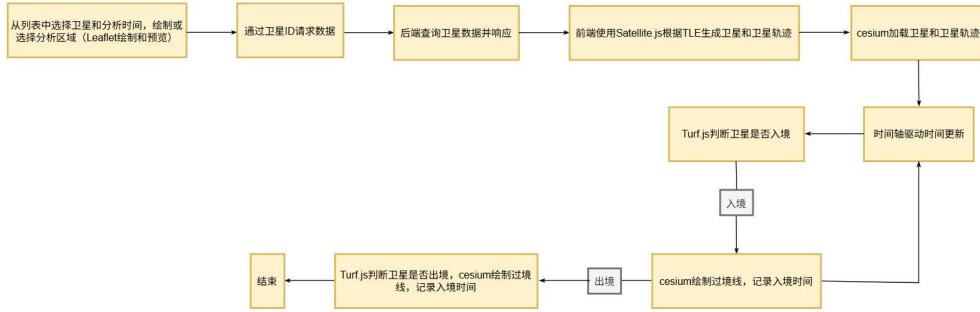


图 3-5 过境分析

(2) 卫星模拟

用户输入长半轴、倾角、偏心率后，Cesium 接收输入参数并生成模拟卫星，然后绘制卫星轨道并显示模拟卫星和轨道。



图 3-6 卫星模拟流程图

(3) 轨迹预测

预测卫星给定时间段中的运行轨迹，用户指定时间区间后，从后端获取指定卫星的 TLE 数据，Satellite.js 根据数据生成卫星轨迹供 Cesium 加载，模拟运行效果。



图 3-7 轨迹预测流程图

(4) 地面站通信模拟

前端请求卫星 TLE 数据，后端响应数据，前端使用 Satellite.js 生成卫星轨迹，供 Cesium 加载，同时 Cesium 加载地面站，设置可通信距离，运行过程中，Cesium 判断卫星和地面站是否在可通行距离内，在可通行距离内则显示通信链路。

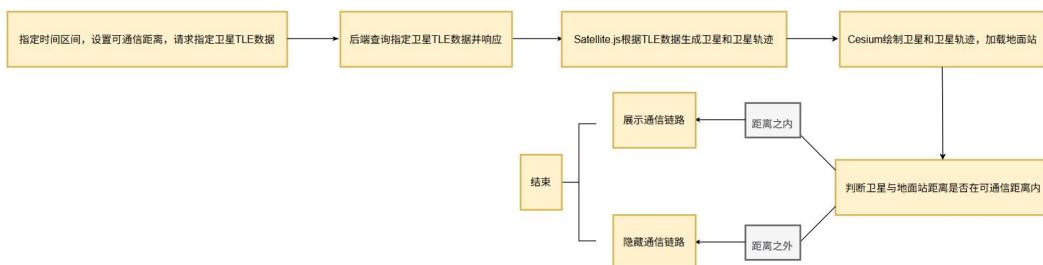


图 3-8 地面站模拟通信流程图

(5) 覆盖分析

用户指定周期，从后端计算指定卫星周期内的星下点并返回，前端获取数据后，Cesium 加载星下点、连点成线，得到覆盖轨迹。

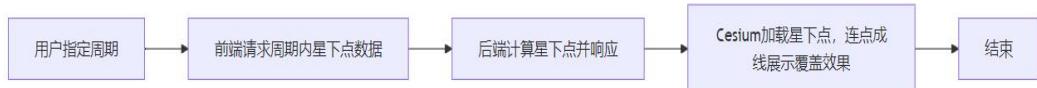


图 3-9 覆盖分析流程图

3.3.4 数据分析统计模块

(1) 数据总览

展示卫星总数、TLE 总数、类型总数、类型种类、卫星所属组织总数、具体组织、卫星轨道类型、具体类型、卫星星座总数、卫星星座类型。

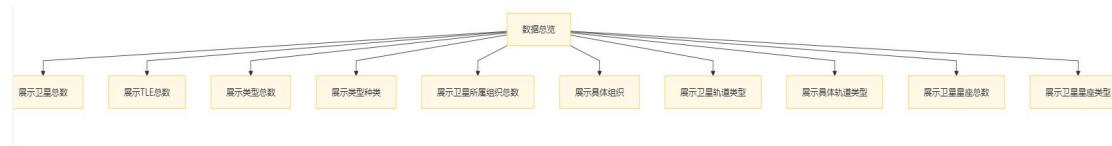


图 3-10 数据总览功能架构

(2) 卫星详细数据

选择卫星，后端返回其基础属性、计算周期、轨道六参数、近地点高度和速度、远地点高度和速度、高度极值、平均高度和高度波动范围等，以及返回该卫星历史 TLE 数据和绘制并返回卫星轨道图片，前端获取数据后进行渲染。

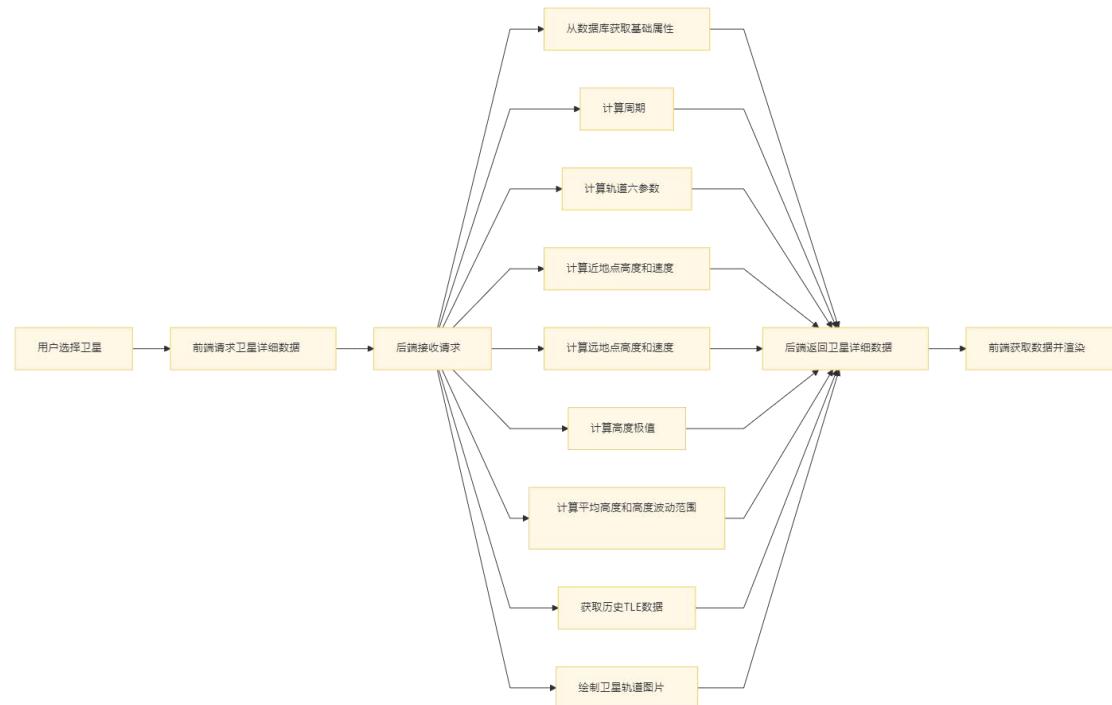


图 3-11 卫星详细数据展示流程图

(3) 卫星类型分类

统计数据库中卫星的类型种类、类型组合（同一卫星可能属多个类型），统计轨道类型种类、轨道类型组合（同一卫星可能属多个轨道类型），前端使用 Echarts.js 生成图表展示。

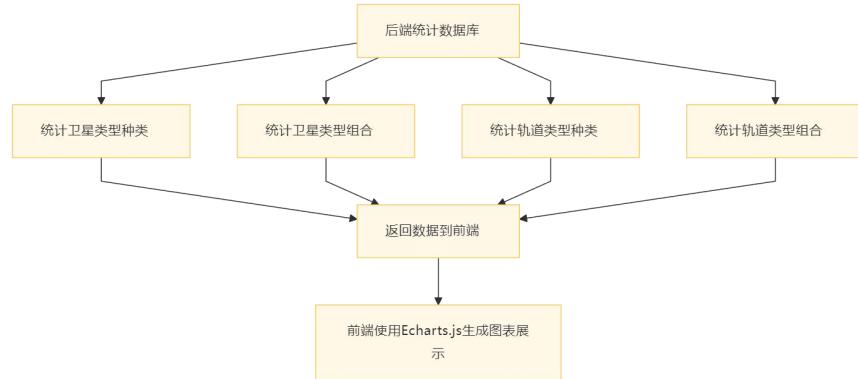


图 3-12 卫星类型分类流程图

(4) 卫星所属分类

统计数据库中卫星的所属国家（组织和地区），前端使用 Echarts.js 生成图表展示。



图 3-13 卫星所属分类流程图

(5) 轨道高度分析

选择卫星，后端计算该卫星高度随时间的变化和随经纬度的变化，返回计算结果给前端，前端绘制图表展示。

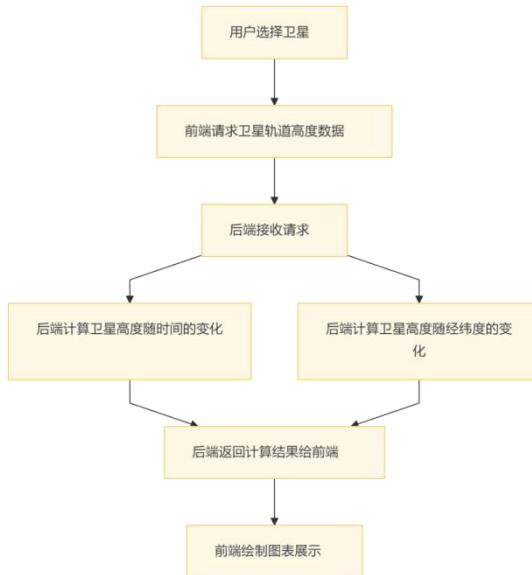


图 3-14 轨道高度分析流程图

(6) 卫星速度分析

选择卫星，后端计算该卫星速度随时间的变化，返回计算结果给前端，前端绘制图表展示。

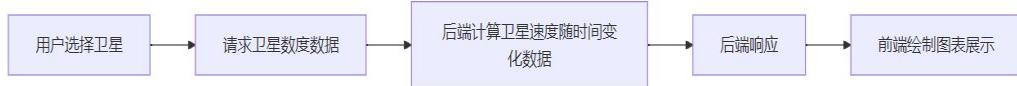


图 3-15 卫星速度分析流程图

(7) 星下点分析

后端计算当前时刻全部卫星的星下点，返回星下点数据给前端，前端使用 Leaflet 绘制星下点分布热力图



图 3-16 星下点分析流程图

(8) 星座分类

后端统计数据库中全部卫星的所属星座和各星座的卫星数量，返回数据给前端，前端绘制柱状图展示。



图 3-17 星座分析流程图

3.3.5 数据管理模块

数据管理采用命令行执行相关数据管理操作，主要分为单记录操作、批量操作、数据导出三模块。

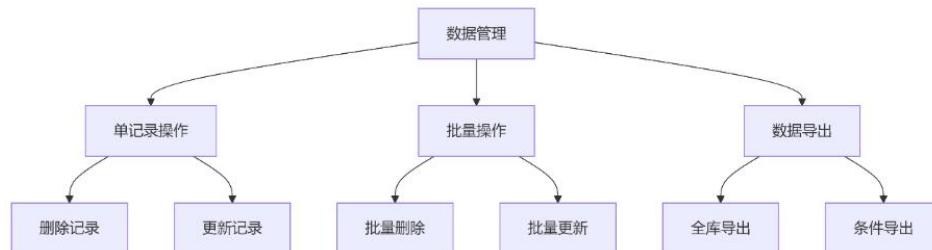


图 3-18 数据管理功能架构图

操作流程为：验证身份进入管理界面，输入命令，解析器解析命令，执行操作、返回结果。

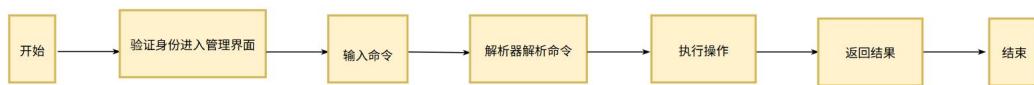


图 3-19 数据管理操作流程图

3.4 图形界面设计

为了全面提升系统的可视化表达能力与用户操作体验,本系统在前端设计中采用了以 Vue 3 为核心框架、PrimeVue 为 UI 组件库的前后端分离架构,并构建了基于 Cesium 与 Spacekit 双引擎融合的可视化模块。图形界面设计整体遵循“清晰的结构布局、统一的视觉风格、友好的交互逻辑与高效的数据反馈”原则,确保系统在科学表达、功能操作和审美体验三方面的平衡统一。

3.4.1 主界面整体布局

系统采用模块化、响应式的页面布局结构,将功能区域划分为五大核心板块:顶部导航栏、左侧功能面板、中央主视图区、底部时间控制器与右侧弹出信息面板。

顶部导航栏展示系统名称与主题切换按钮,同时集成帮助入口、开发者信息等通用功能。左侧控制面板集中管理所有操作入口,包括卫星筛选、轨道模式切换、图层显示控制与模拟器入口,确保用户操作路径明确、逻辑层级清晰。中央主视图区是系统的可视化核心,嵌入了 Cesium 地球渲染模块与 Spacekit 轨道仿真引擎,用户可在同一界面下自由切换或对比不同引擎展示的视角与数据表现。底部时间轴控制条用于调节轨道播放进度、播放速度与时间跳跃,支持动态拖动、暂停、回溯等操作;右侧弹出信息面板则根据不同交互事件自动弹出,包括卫星参数、过境分析、数据图表、实体配置等内容。

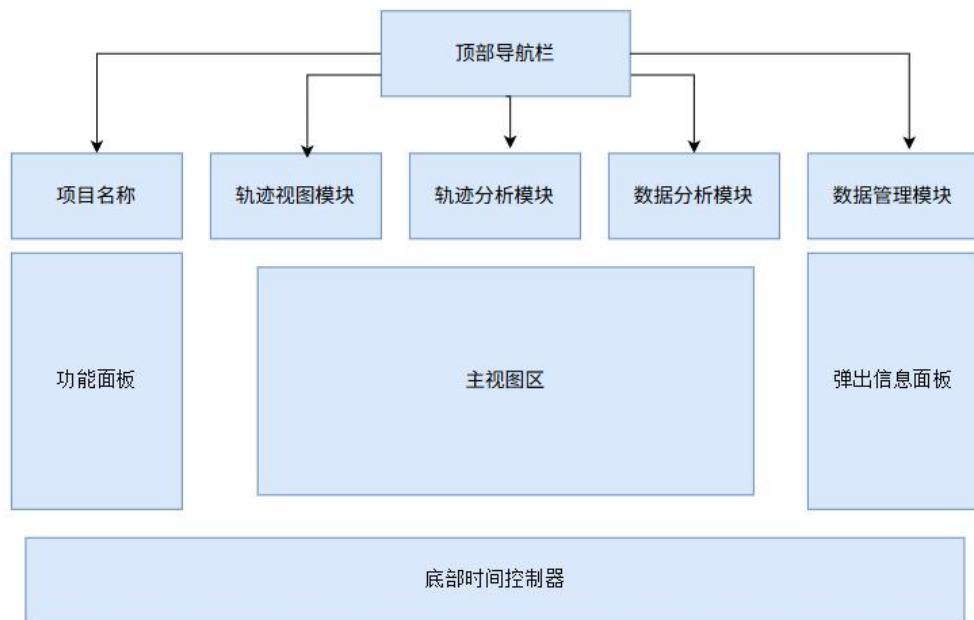


图 3-19 系统主界面布局结构图

3.4.2 双引擎可视化架构集成设计

本系统的核心创新之一在于同时引入了 Cesium 和 Spacekit 两大可视化引擎，并通过统一时间驱动机制实现同步渲染。这一设计既保留了 Cesium 在三维地球与空间数据表达方面的丰富能力，又充分利用了 Spacekit 在轨道物理建模与天体运动仿真方面的精度优势。Cesium 主要负责地理空间渲染，包括地球球体、地表图层、城市模型与卫星轨迹的地面投影，适用于展示星下点、覆盖范围、通信链路等与地球相关的场景。而 Spacekit 则使用 TLE 数据进行轨道推演，构建精确的椭圆轨道模型，并以天文物理的方式渲染轨道运动，展示包括近地轨道、中轨道和高轨道的结构布局。

两者通过统一的时间控制模块进行协同运行，确保在任意时刻下，两个引擎展示的卫星轨迹处于一致的时空状态。用户可以通过界面按钮切换视角，选择以“地球为参考”或“天体为参考”的方式浏览卫星运行轨迹，进一步提升了科学性与趣味性。

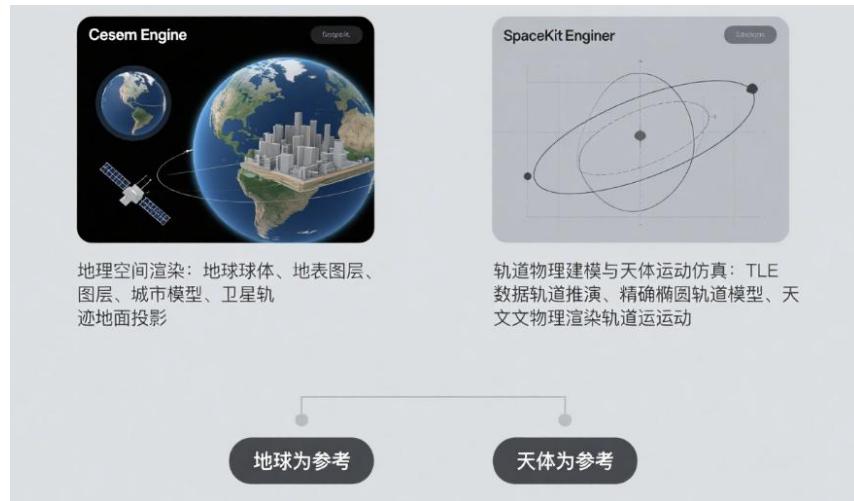


图 3-20 双引擎可视化协同结构图

模块名称	Cesium Viewer	Spacekit Renderer
所属引擎	Cesium	Spacekit
功能职责描述	渲染三维地球、地表图层、地面轨迹投影、星下点、通信视域	精确计算轨道轨迹、渲染物理椭圆轨道、展示轨道结构与天体背景
数据来源 / 输入	ECEF 坐标系轨迹点、图层 GeoJSON 数据	TLE 数据、轨道六参数 $(a, e, i, \omega, \Omega, M)$
输出 / 显示效果	可交互 3D 地球、覆盖区/地面线	空间中真实轨道线、轨道结构图
对接模块	时间轴、图层控制器	时间轴、模拟器

表 3 双引擎可视化协同结构图

3.4.3 图层管理与多视角切换设计

系统在三维空间可视化表达中，引入图层控制机制，用户可按需选择显示或隐藏不同类别信息，包括轨迹线、星下点、通信链路、覆盖范围、地面站点等。所有图层操作均通过统一的 PrimeVue 控制面板进行管理，并使用响应式绑定确保用户修改立即生效。

此外，系统支持多种视角切换操作，包括一键飞行至目标卫星、第一人称卫星视角、地面观察视角与双窗口分屏视图。通过这些视角切换按钮，用户可从多角度理解卫星运行状态，提升对空间态势的直观感知。

3.4.4 样式与组件规范

为了确保前端界面风格统一、用户操作体验良好，系统前端开发严格遵循组件化与规范化设计原则。全部控件基于 PrimeVue 组件库构建，包括按钮（Button）、表格（DataTable）、选择器（Dropdown）、滑块（Slider）、日历（Calendar）等，具备良好的响应性与跨浏览器兼容性。系统配色方案以深蓝色、钛白色为主，辅以橙色高亮指示，整体风格突出科技感与空间感，符合航天类系统的专业定位。

在字体规范上，标题采用加粗黑体，正文字体采用系统默认无衬线字体，字号区分层级明确。布局结构采用模块卡片化封装，每一项功能均具备边框、阴影与操作区域标识，增强模块辨识度。同时，为保障前后端协同开发，所有组件使用统一命名、统一参数格式，并在组件库中封装复用，提升系统可维护性与扩展性。

核心 技术

第四章

空天守望者：人造卫星态势感知系统

第四章 核心技术

4.1 核心技术框架

4.1.1 技术框架

选用 Django 作为后端框架。这是一个开源的高级 Python Web 框架。Python 生态在科学计算和数据可视化方面非常成熟，为轨迹分析提供了有力支持。

(1) 数据处理借助 NumPy 和 Pandas 等库的强大功能，可以高效地处理和操作海量卫星轨迹数据。

(2) Matplotlib 库为生成卫星轨迹及相关数据的可视化图表提供了便利。

(3) sgp4 和 skyfield 等库则简化了复杂的卫星轨道计算任务。

前端采用 Vue3 构建用户界面。它是一个渐进式的 JavaScript 框架。Vue3 能够很好地集成 Cesium.js、Spacekit.js 和 Echarts.js 等强大的可视化库。

(1) Cesium.js 和 Spacekit.js 用于实现逼真的卫星轨迹三维可视化。

(2) Echarts.js 用于生成清晰的轨迹分析图表和数据概览。

4.1.2 可视化技术栈

Cesium 是一个基于 WebGL 的开源 JavaScript 库，用于创建 3D 地球和地理空间数据可视化，支持实时数据展示和交互操作，拥有强大的空间模拟和多源数据兼容能力，能够完成卫星轨迹的可视化和卫星运行模拟。

Spacekit 是一个用于三维空间可视化的 JavaScript 库。提供了丰富的工具和接口，主要用于天文学领域，提供了两个最基本的组件，纹理贴图和动态物理对象。

4.1.3 辅助技术

Turf.js 是一个专注于前端空间分析的 JavaScript 开源库。它提供了丰富的空间计算函数，非常适合与 Cesium.js 结合使用，共同完成高效地理空间分析任务。

Echart.js 是一个基于 JavaScript 的开源图表库。它拥有海量的示例模板，开发者可以快速构建各种类型的数据可视化图表。

Satellite.js 是专为处理卫星轨道数据设计的 JavaScript 开源库。核心价值在于：

(1) 实现了标准的 SGP4/SDP4 算法（需搭配相应的 TLE 数据才能获得最优预报精度），用于计算卫星位置和速度。

(2) 算法实现与 Python 的 sgp4 库保持一致，确保了计算结果的准确性。

(3) 支持多种常用坐标系的转换，例如将地心惯性坐标系（ECI）转换为地

心固定坐标系（ECF）或大地坐标系（LLA）。

(4) 能够直接处理 TLE 数据并生成 Cesium.js 可识别的数据格式，从而在前端高效地加载和展示卫星轨迹。

sgp4 是用于计算卫星轨道的 Python 库，基于 SGP4/SDP4 算法(SGP4/SDP4 在解算时有着运算速度较快的优势，同时能够满足轨道预报所需的精度)。支持从 TLE 或 OMM (Orbit Mean Elements Message) 格式加载卫星轨道参数。

Skyfield 是一个纯 Python 的天文软件包，能够生成行星和地球卫星的高精度位置。支持 TAI、TT、TDB、UT1 等多种时间尺度。可以计算地理坐标和天文位置，支持多种坐标系转换。

4.1.4 数据库技术

PostgreSQL 是一个功能强大的开源关系型数据库管理系统。

Django ORM (Object-Relational Mapping) 是 Django 框架的核心组件之一，用于将 Python 类（模型）映射到数据库表。ORM 允许开发者使用 Python 代码而不是 SQL 语句来操作数据库，使得数据库操作更加直观和安全。

4.2 核心技术相关理论

4.2.1 TLE 数据格式规范

TLE，全称 Two-Line Element（两行元素集），是一种简洁的轨道参数表示方式，用于描述地球轨道上的人造卫星或空间目标的轨道。是目前唯一公开发布且编目最完备的地球轨道空间目标编目数据。TLE 数据通常由两行文字组成，每行包含了一系列轨道参数和标识信息。

(1) TLE 数据解读

TLE 数据示例如图 2-1 所示，包含两行，第一行为基本信息，如卫星编号、国际分类代号、轨道历元（生成时间）。第二行是精确描述卫星的轨道参数，包括倾角、升交点赤经、偏心率等。

● ● ●
1 1 48274U 21035A 22043.0000000 .00015783 00000-0 17678-3 0 00017
2 2 48274 041.4677 261.7593 0006529 279.1126 199.9265 15.61666662045206

图 4-1 TLE 数据示例

两行轨道要素采用两行 80 字符的 ASCII 码来存储数据，美国空军负责跟踪地球轨道上所有可探测物体，并为每个目标生成相应的 TLE 轨道根数，并且把非保密目标的 TLE 数据公布在 Space Track 网站。TLE 轨道根数仅可以表述绕地球运行物体的轨道。

是否保密 (U不 保密, C或S为只 对NORAD开放)	TLE历元时刻 (22年第43天, 即2月12日, 小 数位为该天时间, 发射的物体A, 长五B 遥二一级即为物体B) 即0时 (UTC)										轨道模型参数		轨道模型类型 (0为 SGP4/SDP4轨道模型)	
	1	48274U	21035A	22043.00000000	.00015783	00000-0	17678-3	0	00017	校验位	平均运动	平均运动二阶导数	BSTAR	该物体的 第N组TLE
TLE 行号	2	48274	041.4677	261.7593	0006529	279.1126	199.9265	15.61666662	04520	6校验位	一阶导数 (假设小数点) 每天环绕地球	阻力系数		
											近地点幅角	平近点角		发射以来 的圈数 (其倒数 即为轨道周期)
											升交点赤经	轨道偏心率 (实 际为0.0006529)		
											NORAD编号 (48274为 CSS天和ID)			

图 4-2 TLE 数据詳解

索引	描述	值
01	轨道报行号	1
03—07	卫星目录编号 (即国际代码)	48274
08	轨道根数级别 (即卫星分类, U 代表公开的卫星, 不保密; S 表示秘密, 秘密目标根数不公开)	U
10—11	国际登记号 (航天器发射年份的最后两位)	21
12—14	国际登记号 (发射年份的序号)	035
15—17	国际登记号 (同时发射个数号)	A
19—20	轨道根数纪元 (UTC), 轨道预报的历元年最后 两位	22
21—32	轨道根数纪元 (UTC), 轨道预报的历元时刻 (从 当年的 1 月 1 日起累计天数及当天的分数部分)	043.00000000
34—43	平均运动一阶导数, 计算平均运动变化带来的轨 道位置漂移, 预测卫星位置及校准位置(第 34 字符可 以为'+' / '-' / ' '用于表示导数正负值, 空格为正)	.00015783
45—52	平均运动二阶导数, 用于计算平均运动变化漂移 变化带来的漂移, 用于预测卫星位置和校准位置	00000-0
54—61	BSTAR 阻力扰动系数, 模拟阻力(空气阻力)的系数	17678-3
63	轨道模型类型	0
65—68	星历编号, 按生成的 TLE 数据条数递增	000
69	校验码	17

表 2 TLE 轨道报第一行格式描述表



索引	描述	值
01	轨道报行号	1
03—07	卫星目录编号（即国际代码）	48274
09—16	卫星轨道倾角（°），轨道面和地球赤道的夹角度数， 0-90 度表示逆行轨道(北极看是逆时针), 90-100 度表示逆行 轨道	041.4677
18—25	升交点赤经（°），指卫星由南到北穿过地球赤道平面 时，与地球赤道平面的交角	261.7593
27—33	偏心率（采用十进制小数），椭圆轨道面中心点到地球 球心距离除以轨道半长轴，值为 0~1 的小数，这里只保留小 数部分	0006529
35—42	近地点角距（°），卫星近地点和升交点对地心的张角	279.1126
44—51	平近点角（°），确定卫星经过近地点时间	199.9265
53—63	平均运动周期，即每天绕地球的圈数（转/天）	15.61666662
64—68	纪元轨道飞行圈数，发射以来绕地球飞行圈数	04520
69	校验码	6

表 3 TLE 轨道报第二行格式描述表

(2) TLE 参数的动态更新

TLE 不是固定不变的。卫星轨道会因地球引力变化（地球引力场并不均匀，会导致卫星轨道发生变化）、大气阻力（在低轨道上，卫星受到大气的阻力，导致轨道逐渐衰减）、太阳辐射压力（太阳辐射施加微弱的推动力，影响卫星轨道）、月球和其他天体的引力等因素的影响，卫星轨道会不断变化，超过两周的 TLE 数据将会使预测出现很大偏差，因此需定期更新 TLE 数据，以确保预测的准确性。

(3) TLE 数据的更新

TLE 数据的更新步骤为：数据收集（通过地面雷达和空间跟踪系统收集卫星的实际位置数据）、数据处理（将收集到的数据输入轨道计算模型，如利用 SGP4

更新卫星轨道参数)、数据发布(将更新后的 TLE 数据发布到公开数据库和服务网站,如 celestrak.org,供用户获取最新的 TLE 数据)。

(4) TLE 计算卫星轨道

使用 SGP4/SDP4 轨道预测模型,将 TLE 数据输入,并计算卫星在特定时间在三维空间中的位置(X、Y、Z 坐标)和速度(Vx、Vy、Vz)。

4.2.2 SGP4/SDP4 轨道预测模型

NORAD 将空间目标分为近地目标(轨道周期小于 225 min)和深空目标(轨道周期大于或等于 225 min)两类。根据轨道周期的不同,可选择与 TLE 相对应的近地或深空计算模型。

SGP4 模型由 Hilton Kuhlman 于 1966 年开发,适用于近地目标的预报。它简化了古在由秀 1959 年提出的引力场模型,认为大气阻力对平动的影响随时间呈线性变化,平近点角的摄动项是时间的二次函数,且阻力对偏心率的影响使近地点的高度保持常值。SGP4 模型由 Ken Cranford 于 1970 年开发,适用于近地目标。模型根据 Brouwer 的引力场模型和 Lane 的大气密度模型对推出的解析公式做了简化。SDP4 模型是 SGP4 模型的外延,适用于深空目标。

4.2.3 轨道参数

轨道参数(或称轨道要素或轨道根数)是描述在牛顿运动定律和牛顿万有引力定律的作用下的天体或航天器,在其开普勒轨道上运动时,确定其轨道所必要的六个参数。知道这六个参数就可以知道和预测航天器的位置和轨道。各参数义如表 3 所示,各参数空间关系如图 4-3 所示。

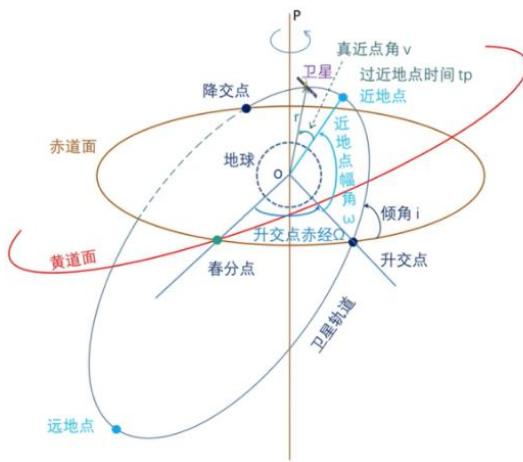


图 4-3 轨道根数空间关系

符 号	名称	定义	备注
a	半长轴	轨道椭圆长轴的一半，决定轨道大小。	单位通常为米 (m) 或千米 (km)
e	偏心率	描述卫星轨道面形状(椭圆扁率)和偏离理想中心位置	$0 \leq e < 1$ (椭圆轨道)； $e=0$ 时为圆轨道
i	轨道倾角	轨道平面与地球赤道平面的夹角	角度 (或弧度)
Ω	升交点赤经	春分点方向到升交点 (卫星向北穿越赤道的位置) 的夹角。	角度 (或弧度)
ω	近地点幅角	升交点到近地点 (轨道离地球最近点) 的夹角。	角度 (或弧度)
v	真近点角	卫星当前在轨道上的位置相对于近地点的角度	角度 (或弧度)

表 4 轨道六参数表

其中表 4-3 “近地点幅角”和表 4-2 中“近地点角距”是等价的，二者描述的是同一个概念。需要注意的是上述两表中，平近点角 (M) 和真近点角 (v) 的区别：

- (1) TLE 中直接提供的是平近点角 (M)，而非真近点角 (v)。
- (2) 平近点角 (M) 描述卫星在参考时间 (历元时间) 的虚拟角度，真近点角 v 反映了航天器在轨道上相对于近地点的瞬时位置。
- (3) 平近点角 M 与时间的关系是线性的 (由近点起算, 随时间均匀变化, 其变率为椭圆运动的平均角速度 n , 故为时间的线性函数)，可通过平近点角 M 计算出偏近点角 E ，再通过偏近点角 E 计算出真近点角 v 。即可确定卫星具体位置。

4.2.4 卫星轨道分类

卫星飞行的水平速度叫第一宇宙速度，即环绕速度。卫星只要获得这一水平方向的速度后，不需要再加动力就可以环绕地球飞行。这时卫星的飞行轨迹叫卫星轨道。太空中的卫星在地球引力等各种力的作用下做周期运动，一阶近似就是一个开普勒椭圆轨道。卫星轨道的形状和大小是由长轴和短轴决定的，而交点角 Ω 、近地点幅角 ω 和轨道倾角 i 则决定轨道在空间的方位。

根据不同的用途，卫星会选择不同的轨道，表 4 介绍常见的轨道类型及分类依据。

轨道类型	缩写	关键判断条件	典型参数范围
地球静止轨道	GEO	周期≈1436分钟(24小时) 高度≈35786km 倾角<5° 偏心率<0.01	单位通常为米(m)或千米 (km)
倾斜地球同步轨道	IGSO	周期≈1436分钟 倾角>5° 偏心率<0.01	倾角: 5°-30° 高度同 GEO
地球同步转移轨道	GTO	近地点<1000km 远地点>35000km 偏心率 0.6-0.8	周期: 5-16 小时
高椭圆轨道	HEO	偏心率>0.1	远地点高度>40000km
低地球轨道	LEO	平均高度<2000km	高度: 160-2000km 周期: 87-127 分钟
中地球轨道	MEO	2000km≤高度<35786km	高度: 2000-35786km 周期: 2-24 小时
太阳同步轨道	SSO	倾角 96°-100° 高度<2000km	倾角: 97°-99° 高度: 600-800km

表 5 轨道分类表

轨道类型	缩写	关键判断条件	典型参数范围
闪电轨道	MO	倾角≈63.4° 远地点>35000km 周期≈718 分钟(12 小时)	近地点: 400-600km 远地点: 38000-40000km
赤道轨道	EO	倾角≈0°	倾角<1°
极地轨道	PO	倾角 85°-95°	倾角≈90°
逆行轨道	AO	倾角 0°-90°	-
逆行轨道	RO	倾角 90°-180°	-

续表 4 轨道分类表

创新特色

第五章

空天守望者：人造卫星态势感知系统

第五章 创新特色

本系统在面对传统卫星轨迹可视化系统存在的精准度不足、实时性较差、交互性较弱、学习成本高等问题基础上，从架构设计、数据处理、可视化引擎、交互逻辑与系统性能等多个维度提出并实践了创新解决方案，形成以下四项核心创新特色：

5.1 架构融合创新：Cesium + Spacekit 双引擎协同机制

问题背景：

传统可视化平台通常仅使用单一渲染引擎，难以同时兼顾地理空间信息的真实感表达与天文轨道物理的精准建模，导致轨道显示失真或缺乏三维立体感。

创新点：

系统提出“Cesium + Spacekit”双引擎协同架构。Cesium 用于地球表面渲染与空间定位，Spacekit 负责卫星轨道轨迹模拟与物理运动推演。

技术实现：

1. 使用时间同步模块协调 Cesium 与 Spacekit；
2. 使用 Vue 组件化机制封装双引擎加载逻辑，确保渲染时序一致；
3. 通过统一时间轴组件对两套引擎统一驱动，实现轨迹同步播放。

应用价值：

该机制突破了单引擎性能与精度的限制，使系统同时具备真实地球仿真与物理轨道准确性，视觉效果与科学性兼备。

5.2 数据管理创新：TLE 版本控制与动态更新机制

问题背景：

TLE 数据随时间不断变化，常规系统缺乏有效的轨道数据管理手段，易导致轨道计算偏差或数据冗余。

创新点：

系统引入 TLE 版本化管理机制，通过对历元时间实现自动更新，并将旧版本 TLE 归档管理。

技术实现：

1. 引入“卫星主表 + 历史 TLE 分区表”数据库设计，使用 PostgreSQL 哈希分区提升查询效率；
2. 实现数据更新幂等性判断与行级锁定控制，防止并发冲突；

3.每次更新后自动归档旧 TLE，实现轨道变化过程可追溯。

应用价值：

实现轨道数据生命周期管理，增强系统的鲁棒性和科学分析基础，为后续轨道精度对比与异常识别提供数据支撑。

5.3 坐标系处理创新：ECI-ECEF 高精度实时转换模块

问题背景：

卫星轨道本体位于地心惯性坐标系（ECI），而三维地球展示通常基于地心固定坐标系（ECEF），两者之间的误差会严重影响轨迹可视化的精度。

创新点：

系统自主开发实时坐标转换模块，实现 ECI 到 ECEF 的动态转换，确保轨道与地球渲染的一致性。

技术实现：

- 1.利用 Skyfield 与 SGP4/SDP4 模型获取 ECI 轨道位置；
- 2.基于时间戳计算地球自转角度，将 ECI 向 ECEF 旋转映射；
- 3.补偿地球形变与椭球建模误差，增强对极轨/斜轨卫星轨迹的拟合精度。

应用价值：

解决轨道漂移和错位问题，保障三维轨迹精准叠加，支撑过境分析与通信模拟等依赖高空间精度的功能模块。

5.4 交互逻辑创新：空间-时间双维度可视化控制

问题背景：

多数轨道可视化平台只能播放静态轨迹或缺乏交互维度，用户无法主动定义分析区域或动态查询指定时刻信息，限制了用户参与深度。

创新点：

系统设计了融合空间（地理围栏）与时间（统一时间轴）的双维度控制机制，实现轨迹实时调整、区域过境分析、星下点可视化等动态交互功能。

技术实现：

- 1.时间轴组件支持自定义播放速率、时间跳转、暂停与反向播放；
- 2.地理围栏通过 Turf.js 实现空间内点判断，驱动过境分析逻辑；
- 3.点击轨迹线返回过境/出境时间，结合弹出面板增强信息表达。

应用价值：

提升用户对轨道运行状态的理解与探索能力，使系统从被动浏览平台转变为主动分析工具，适用于科研教学与业务辅助等场景。

系统功能实现

第六章

空天守望者：人造卫星态势感知系统

第六章 系统功能实现

6.1 建库与数据爬取和更新

6.1.1 建库

为实现卫星数据的存储、管理与历史追溯，选用 PostgreSQL 关系型数据库。

根据设计，实现卫星数据表（Satellites）用于存储卫星基础信息及轨道参数，支持多维度查询（如按国家、轨道类型分类），并为历史 TLE 数据提供外键关联。实体关系图如图 6-1 所示，SQL 语句如图 6-2 所示。

satellites[satellites]			
<PK>	id	VARCHAR	50
	name	VARCHAR	255
	inid	VARCHAR	50
	constellation	VARCHAR	100
	year	int2	
	ordinal	VARCHAR	50
	country	VARCHAR	100
	type	_text	
	tracktype	_text	
	line1	VARCHAR	100
	line2	VARCHAR	100
	epoch	timestamp	

图 6-1 satellites 表实体关系图

```
CREATE TABLE release.satellites (
    id          VARCHAR(50)   PRIMARY KEY NOT NULL,
    name        VARCHAR(255)  NOT NULL,
    inid        VARCHAR(50)   UNIQUE NOT NULL,
    constellation VARCHAR(100),
    year        SMALLINT,
    ordinal     VARCHAR(50),
    country     VARCHAR(100),
    type        TEXT[],
    tracktype   TEXT[],
    line1       VARCHAR(100),
    line2       VARCHAR(100),
    epoch       TIMESTAMP
);
```

图 6-2 satellites 表建表语句

(1) 主键设计

采用业务主键 id（NORAD ID）而非自增 ID，避免因数据迁移导致的主键冲突，直接关联国际通用标识。

(2) 数组类型字段

type 和 tracktype 使用 TEXT[]数组类型，支持高标签检索与聚合统计。

(3) 时间精度控制

epoch 字段定义为 TIMESTAMP WITHOUT TIME ZONE，明确约定所有时间以 UTC 标准存储，避免时区转换误差。

实现历史 TLE 数据表 (historytle)，实体关系图 (ER Diagram) 如图 6-3 所示，SQL 语句如图 6-4、6-5 所示。

historytle[historytle]		
<PK>	id	serial
	satelliteid	VARCHAR(50)
	line1	VARCHAR(100)
	line2	VARCHAR(100)
	epoch	timestamp

图 6-3 historytle 表实体关系图

```
CREATE TABLE release.historytle (
    id SERIAL PRIMARY KEY,
    satelliteid VARCHAR(50) NOT NULL,
    line1 VARCHAR(100),
    line2 VARCHAR(100),
    epoch TIMESTAMP,
    FOREIGN KEY (satelliteid) REFERENCES release.satellites(id)
);
```

图 6-4 historytle 表建表语句

```
CREATE TABLE release_history_0 PARTITION OF release_history
FOR VALUES WITH (MODULUS 4, REMAINDER 0);

CREATE TABLE release_history_1 PARTITION OF release_history
FOR VALUES WITH (MODULUS 4, REMAINDER 1);

CREATE TABLE release_history_2 PARTITION OF release_history
FOR VALUES WITH (MODULUS 4, REMAINDER 2);

CREATE TABLE release_history_3 PARTITION OF release_history
FOR VALUES WITH (MODULUS 4, REMAINDER 3);
```

图 6-5 historytle 表分区语句

技术实现要点为使用外键级联策略，通过 FOREIGN KEY (satelliteid) REFERENCES satellites(id) 实现级联删除，当卫星数据被移除时，其历史 TLE 自动清理；使用哈希分区策略，通过 satelliteid 字段进行分区，因为查询历史 TLE 数据总是要先确定所要查询的卫星，因此可以通过 satelliteid 直接定位到相关的分区，从而避免扫描整个表。

图 6-6 展示了卫星数据表 (satellites) 与历史 TLE 表 (historytle) 的关联关系。

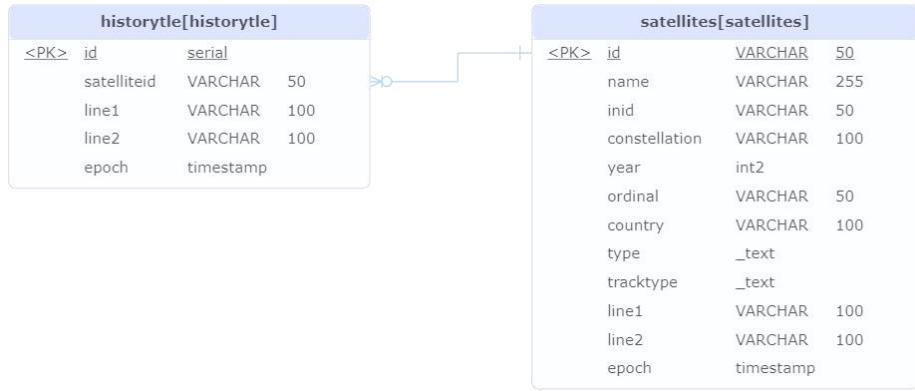


图 6-6 数据表实体关系图

6.1.2 数据爬取

代码主要实现从 celestrak.org (北美空防司令部创建的公益网站, 提供最新的 TLE 数据) 获取卫星 TLE (两行轨道元素) 数据, 解析并计算轨道参数, 最终存储到 PostgreSQL 数据库。核心流程分为:

(1) 数据爬取

从多个预定义 URL 获取 TLE 数据, 按每 3 行解析 TLE 数据, 跳过名称含 R/B 的废弃卫星。

```

FUNCTION get_data_first(url, type):
    SEND HTTP GET 请求到 url
    IF 响应失败: RETURN 空列表

    raw_data = 响应文本按行分割
    satellites = []

    FOR 每 3 行 IN raw_data:
        name, line1, line2 = 组内三行
        IF name 包含 "R/B": 跳过 (废弃卫星)

        # 提取卫星编号和标识符
        satellite_id = line1[2:7] # NORAD ID
        international_id = line1[9:16]

        # 解析发射年份和序号
        IF 国际标识符匹配正则:
            year = 转换年份 (如 22 → 2022)
            ordinal = 发射序号 (如 003A)

        # 计算轨道元数据
        orbit_metadata =
            analyze_altitude_variation(line1, line2)
        tracktype = orbit_metadata["orbit_types"]

        # 构建卫星对象
        satellite = {
            "type": type,
            "name": name,
            "satellite_id": satellite_id,
            "tracktype": tracktype,
            ...
        }
        ADD satellite 到 satellites 列表

    RETURN satellites

```

图 6-7 数据爬取入口函数

(2) 数据解析

通过数学计算和天文库（skyfield）解析 TLE 数据与轨道计算，分析卫星在指定时间段内的轨道高度变化，解析 TLE 获取轨道周期，用于后续推断其轨道类型。

```
FUNCTION analyze_altitude_variation(line1, line2):
    初始化卫星对象 (使用 Skyfield 库)
    解析 TLE 获得轨道参数 (调用 tle_to_orbital_elements)

    # 生成时间序列 (默认 24 小时)
    start_time = 当前时间
    time_points = 每隔 step_min 分钟生成时间点

    # 计算每个时间点的高度
    FOR 每个时间点 t IN time_points:
        altitude = 卫星在 t 时刻的地面高度 (km)

    # 计算平均高度和轨道类型
    mean_altitude = 所有高度的平均值
    orbit_types = 调用 determine_orbit_type 判断类型
    RETURN 包含轨道元数据的 JSON
```

图 6-8 数据解析核心代码

从 TLE 第二行 (line2) 中提取轨道六要素，用于后续计算。

```
FUNCTION tle_to_orbital_elements(line2):
    # 从 TLE 第二行提取轨道六要素
    inclination = line2[8:16] 解析为浮点数      # 轨道倾角 (度)
    raan = line2[17:25] 解析为浮点数            # 升交点赤经 (度)
    eccentricity = "0." + line2[26:33] 解析为浮点数 # 偏心率 (0.xxxx)
    arg_perigee = line2[34:42] 解析为浮点数      # 近地点幅角 (度)
    mean_anomaly = line2[43:51] 解析为浮点数      # 平近点角 (度)
    mean_motion = line2[52:63] 解析为浮点数      # 平均运动 (转/天)
    RETURN inclination, raan, eccentricity, arg_perigee, mean_anomaly, mean_motion
```

图 6-9 提取轨道六要素

(3) 轨道分析

通过数学计算和天文库（skyfield）分析卫星轨道类型。通过轨道周期、倾角、偏心率等参数，依据 4.2.4 卫星轨道分类中表 4 标准，判断卫星类型（如 GEO、LEO 等）。

```
FUNCTION determine_orbit_type(inclination, eccentricity,
...):
    IF 轨道周期接近24小时 (GEO条件) :
        ADD "GEO" 或 "IGSO" 到类型列表
    ELSE IF 偏心率 > 0.1:
        ADD "HEO" 到类型列表
    ELSE:
        IF 平均高度 < 2000 km: ADD "LEO"
        ELSE IF 2000 ≤ 高度 < 35786 km: ADD "MEO"
        ELSE: ADD "HEO"

    IF 倾角接近极地范围 (80°~180°) : ADD "PO"
    IF 倾角接近太阳同步轨道 (96°~100°) : ADD "SSO"
    RETURN 轨道类型列表
```

图 6-10 轨道分类

(4) 数据库存储

将处理后的数据插入或更新到数据库。

```
● ● ●  
FUNCTION insert_satellite_into_db(satellite):  
    CONNECT 到 PostgreSQL 数据库  
  
    # 尝试插入新记录 (NORAD ID 为主键)  
    EXECUTE SQL:  
        INSERT INTO satellites (...)  
        VALUES (...)  
        ON CONFLICT (id) DO NOTHING  
  
    IF 插入失败 (主键冲突):  
        # 获取现有记录的 type 数组  
        EXECUTE SQL: SELECT type FROM satellites WHERE  
        id = ?  
        existing_types = 查询结果  
  
        IF 当前卫星类型不在 existing_types 中:  
            new_types = existing_types + [当前类型]  
            EXECUTE SQL: UPDATE satellites SET type =  
            new_types WHERE id = ?  
  
    CLOSE 数据库连接
```

图 6-11 数据插入

流程图如图 6-12 所示，数据爬取时序图如图 6-13 所示：

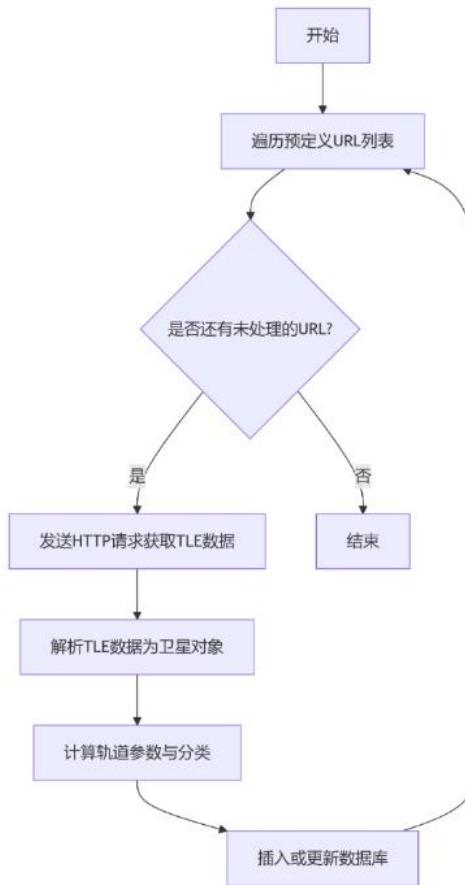


图 6-12 数据爬取流程图

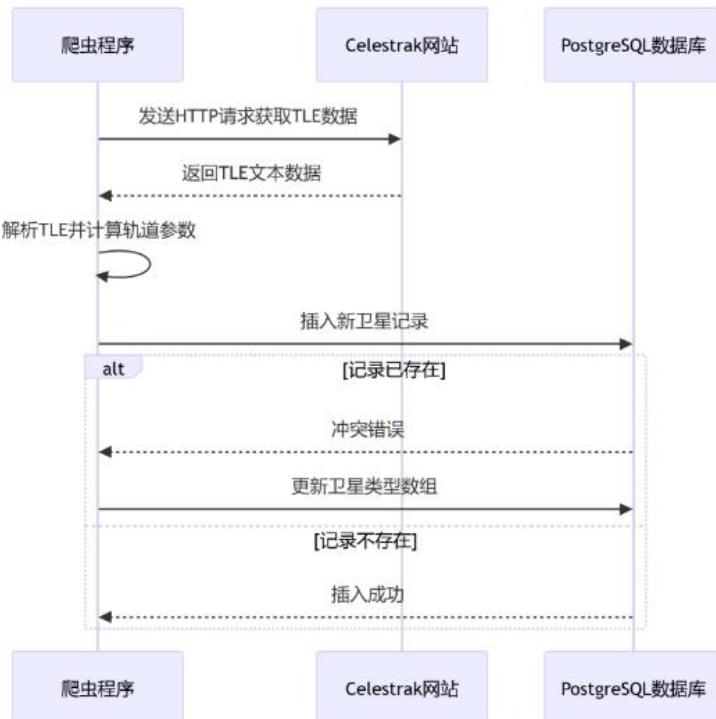


图 6-13 数据爬取时序图

由于 TLE 数据不反映卫星所属星座和所属国家（组织或地区），因此在数据爬取完成后，使用 SQL 语句继续卫星所属星座信息的提取（同一星座的卫星名称中会体现其所属星座），再根据星座信息为依据，查询其所属国家（组织或地区）。

```

● ● ●

-- 更新 constellation 字段为 name 字段的相同部分
UPDATE release.satellites
SET constellation = regexp_replace(name, '^(.* -).*', '\1');

-- 根据 constellation 字段分组，聚合 name 字段
SELECT
    constellation,
    array_agg(name) AS names
FROM
    release.satellites
GROUP BY
    constellation;

```

图 6-14 SQL 语句获取卫星星座

6.1.3 数据更新

数据更新核心功能是从 Celestrak 抓取最新卫星 TLE 数据，更新到 PostgreSQL 数据库，数据流控制为主函数（图 5-17）通过嵌套循环遍历：分类、URL、卫星数据三层结构。数据更新逻辑流程如图 5-16 所示，逻辑间时间关系如图 5-15 所示。

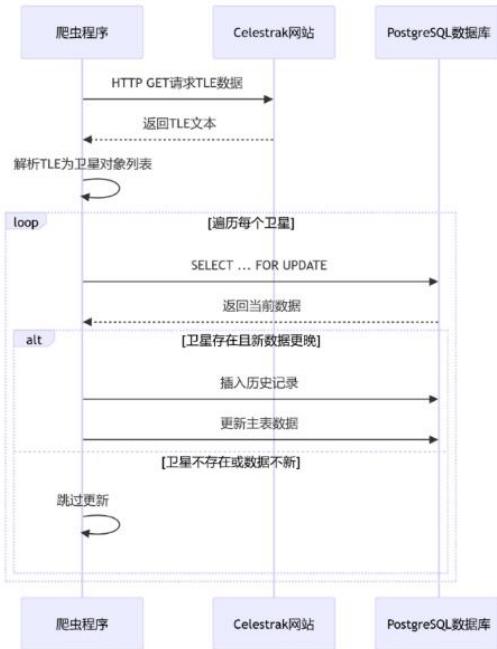


图 6-15 数据更新时序图

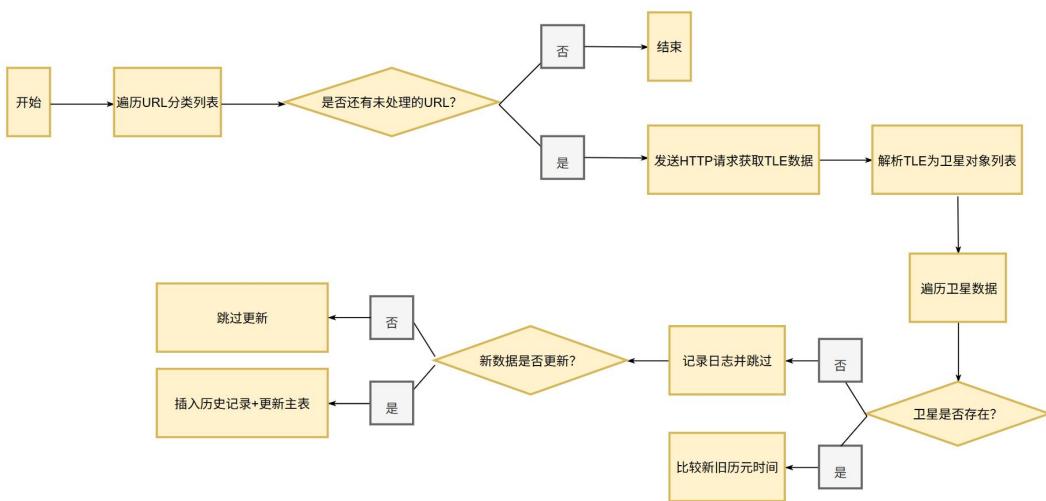


图 6-16 数据更新流程图

具体数据更新流程及关键代码逻辑如下：

(1) 爬取与解析数据

`update_tle_data` 函数为入口函数，实现从预定义的分类 URL 列表下载 TLE 数据。提取卫星编号、轨道参数、历元时间等关键信息。`get_data_first` 函数是实现爬取数据与解析的核心函数，其逻辑为提取 TLE 数据并解析，计算历元时间。



```

● ● ●
FUNCTION update_tle_data():
    # 预定义分类URL列表
    urls = [
        {type: "spacestation", urls: [...]},
        {type: "environment", urls: [...]},
        ...
    ]

    # 遍历所有分类URL
    FOR url_group IN urls:
        PRINT "正在处理分类: {url_group.type}"

        # 遍历组内每个URL
        FOR url IN url_group.urls:
            PRINT "爬取URL: {url}"

            # 获取并解析TLE数据
            satellites = get_data_first(url,
                url_group.type)

            # 遍历每个卫星数据
            FOR satellite IN satellites:
                # 更新数据库
                updated = update_satellite_data(
                    conn,
                    satellite_id =
                satellite["satellite_id"],
                    new_line1 = satellite["line1"],
                    new_line2 = satellite["line2"],
                    new_epoch = satellite["epoch"]
                )

                # 处理更新结果
                IF updated is None:
                    PRINT "卫星ID不存在"
                ELIF updated:
                    PRINT "数据已更新"
                ELSE:
                    PRINT "数据无需更新"

                SLEEP 70秒 # 反爬策略

```

图 6-17 update_tle_data 函数

```

● ● ●
FUNCTION get_data_first(url, type):
    TRY:
        response = HTTP.GET(url) # 发送请求
        response.raise_for_status()
    EXCEPT:
        PRINT "请求失败"
        RETURN []

    # 分割TLE文本
    raw_lines = response.text.split("\n")
    satellites = []

    # 按3行一组处理TLE
    FOR i IN 0 TO len(raw_lines) STEP 3:
        name, line1, line2 = raw_lines[i:i+3]

        IF name包含"R/B": # 过滤废弃卫星
            CONTINUE

        # 提取关键标识
        satellite_id = line1[2:7] # NORAD ID
        international_id = line1[9:16] # 国际编号

        # 解析发射年份
        IF 正则匹配成功(line1, "\d{5}U (\d{2})(\d{3}[A-Z]?)"):
            launch_year = 匹配结果[0]
            full_year = 转换年份(launch_year) # 如22→2022

        # 计算历元时间
        satellite_obj = EarthSatellite(line1, line2)
        epoch = satellite_obj.epoch.utc时间转字符串()

        # 构建数据对象
        ADD {
            "satellite_id": satellite_id,
            "line1": line1,
            "line2": line2,
            "epoch": epoch
        } TO satellites

    RETURN satellites

```

图 6-18 get_data_first 函数

(2) 更新数据库

对比新旧数据的历元时间，仅更新更新的数据，并维护历史记录。时间比较通过 compare_time 函数实现，用于比较 TLE 的历元时间，判断是否执行更新操作，数据库更新逻辑通过 update_satellite_data 函数实现，进行数据更新操作。

```

● ● ●
FUNCTION compare_time(time1_str, time2_str):
    # 统一转为datetime对象
    date_format = "%Y-%m-%d %H:%M:%S.%f"
    time1 = datetime.strptime(time1_str, date_format)
    time2 = datetime.strptime(time2_str, date_format)

    RETURN time1 > time2 # 新数据是否更晚

```

图 6-20 compare_time 函数

```

● ● ●
FUNCTION update_satellite_data(conn, satellite_id,
new_line1, new_line2, new_epoch):
    TRY:
        # 开启事务
        conn.autocommit = False

        # 查询当前数据并锁定行
        cursor.execute("""
            SELECT line1, line2, epoch
            FROM satellites
            WHERE id = %s
            FOR UPDATE
        """, (satellite_id,))
        current_data = cursor.fetchone()

        IF current_data为空: # 卫星不存在
            PRINT "ID不存在"
            RETURN None

        # 解锁旧数据
        old_line1, old_line2, old_epoch = current_data

        # 比较时间戳
        IF compare_time(new_epoch, old_epoch):
            # 归档旧数据到历史表
            cursor.execute("""
                INSERT INTO historytle
                (satelliteid, line1, line2, epoch)
                VALUES (%s, %s, %s, %s)
            """, (satellite_id, old_line1, old_line2,
old_epoch))

            # 更新主表
            cursor.execute("""
                UPDATE satellites
                SET line1=%s, line2=%s, epoch=%s
                WHERE id=%s
            """, (new_line1, new_line2, new_epoch,
satellite_id))

            conn.commit() # 提交事务
            RETURN True
        ELSE:
            conn.rollback() # 回滚事务
            RETURN False

    EXCEPT:
        conn.rollback()
        RAISE错误
    FINALLY:
        conn.autocommit = True

```

图 6-19 update_satellite_data 函数

上述设计通过如下方法，实现了数据爬取的安全性和高效性。

- (1) 实现数据幂等性，仅当新数据的历元时间更新时，才更新主表。
- (2) 实现行级锁定，通过 FOR UPDATE 锁锁定当前卫星记录和历元时间对比，确保数据更新不会重复，防止出现并发冲突。
- (3) 实现版本管理，每次更新前将旧数据归档到 historytle 表，便于追溯。
- (4) 实现错误隔离，单条卫星更新失败不影响整体任务，事务粒度控制在单条记录。

6.2 开发环境搭建

6.2.1 后端开发环境

- (1) 安装 Python 3.12.0。
- (2) 安装 Django 5.0.2。
- (3) 创建项目，进行项目初始化配置。
- (4) 配置数据库，并进行 ORM 模型的创建。



6.2.2 前端开发环境

- (1) 安装 Node.js 和 npm。
- (2) 安装 Vue、Vite、Vue CLI 然后创建项目。
- (3) 安装依赖: vue-router、pinia、axios、cesium、leaflet、turf、echarts、three、Spacekit 等。
- (4) 进行项目配置。
- (5) 进行基础工具函数的封装

6.3 核心功能实现

6.3.1 时间控制组件

时间控制组件依赖 Timeline.js 实现，模块关系如图 6-21，包含以下功能：

- (1) 时间轴可视化与拖拽定位；
- (2) 时间流速控制（加速/减速/恢复）；
- (3) 播放/暂停状态切换；
- (4) 与 Cesium/Spacekit 可视化引擎的时间同步。

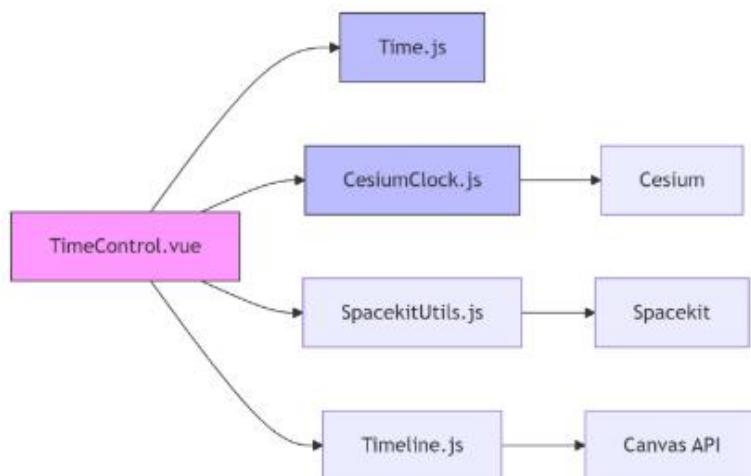


图 6-21 模块关系图

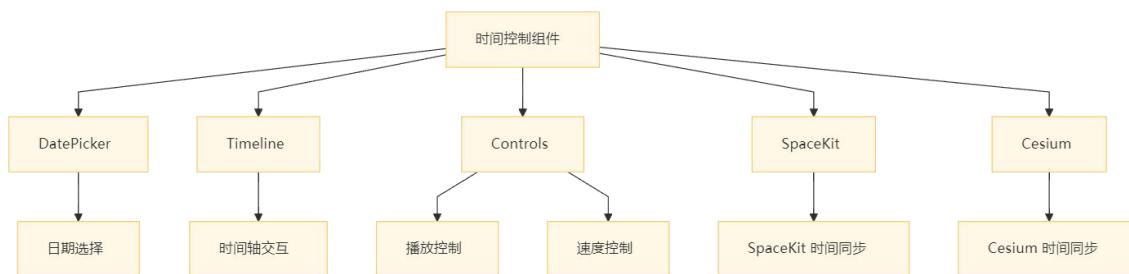


图 6-22 时间控制组件功能逻辑

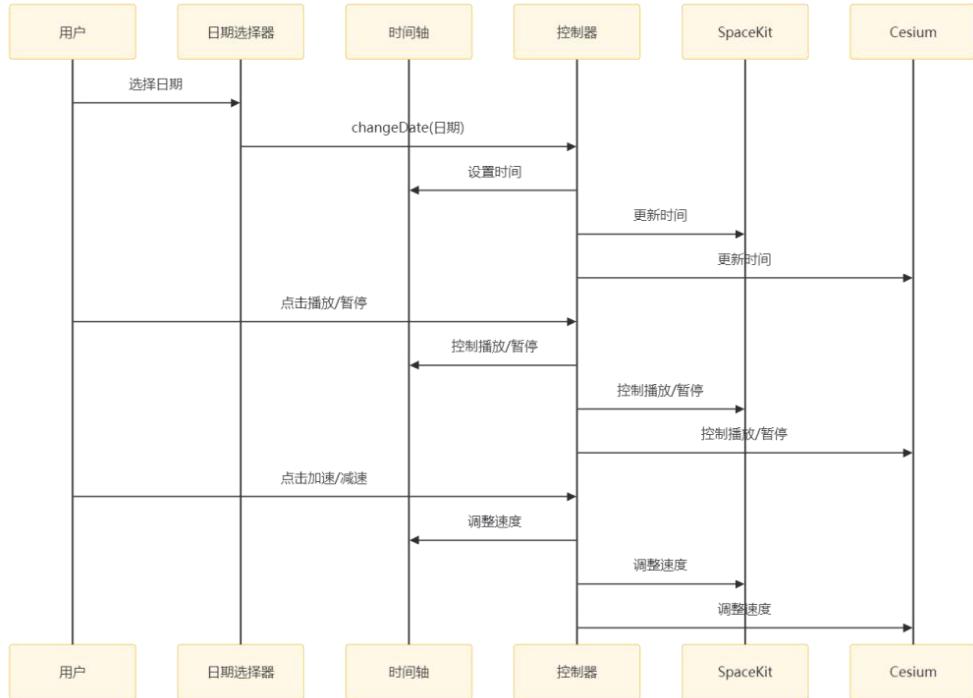


图 6-23 时间控制组件时序图

时间控制相关逻辑如伪代码所示(图 6-24):

```


    定义 controls 对象{
        方法 setSpeed(speed){
            timeSpeed = speed * 1000 // 设置速度 (建议范围 ±1000)
        }
        方法 play(){
            isPlaying = true
            如果 (!isPlaying){
                lastTimestamp = 当前时间戳 // 使用 performance.now() 获取当前时间戳
                调用 animate() // 开始动画
            }
            如果 (props.father == "spacekit"){
                spaceKitStore.viz.start() // 启动 Spacekit 可视化
                // spaceKitStore.viz.setDpsPerSecond(2) // 设置每秒的帧数以增量
                spaceKitStore.viz.removeObject(spaceKitStore.earth) // 移除旧的地球对象
                // 重新创建地球对象
                spaceKitStore.earth = SpacekitUtils.createEarth(spaceKitStore.viz, timeSpeedKnob.value / 86400)
            }
            否则{
                CesiumClock.play(cesiumStore.viewer.clock) // 启动 Cesium 时钟
                // cesiumStore.viewer.clock.shouldAnimate = true // 启用动画
            }
        }
        方法 stop(){
            isPlaying = false
            如果 (props.father == "spacekit"){
                spaceKitStore.viz.stop() // 停止 Spacekit 可视化
                spaceKitStore.viz removeObject(spaceKitStore.earth) // 移除旧的地球对象
                // 重新创建地球对象
                spaceKitStore.earth = SpacekitUtils.createEarth(spaceKitStore.viz, 0)
            }
            否则{
                CesiumClock.pause(cesiumStore.viewer.clock) // 停止 Cesium 时钟
            }
        }
        方法 jumpTo(time){
            currentUnixTime = time // 设置当前时间
        }
        方法 getTimeSpeed(){
            返回 timeSpeed / 1000 // 返回当前速度
        }
    }
}


```

图 6-24 时间控制逻辑伪代码

实现要点主要包括双引擎适配、性能优化、时空约束、状态同步和动态速率:

- (1) 双引擎适配: 通过 props.father 参数区分 Cesium/Spacekit 实现差异;
- (2) 性能优化: 采用 requestAnimationFrame 实现流畅动画;
- (3) 时空约束: 边界检查确保时间在[minTime, maxTime]范围内;

(4) 状态同步:

Cesium 时间同步相对简单，通过 Cesium.Clock 实现，提供了丰富接口实现时间范围设置、时间流速控制、场景时间同步功能。

Spacekit 时间同步较为复杂，在 Spacekit 中，时间单位是 Julian Day（儒略日），每过一秒钟，场景中的时间推进一天。前端时间单位是毫秒。需要将前端时间戳（毫秒）转换为 Julian Day。将真实时间映射到模拟天文时间，针对时间和速度进行如下换算：

① 时间换算：

$$JulianDay = \frac{\text{时间戳 (毫秒)}}{86400000} + 2440587.5 \quad (6-1)$$

注：1970-01-01 午夜（Unix 起点）的儒略日是 2440587.5

② 速度换算：

$$JulianDay/s = \frac{\text{速度 (毫秒每秒)}}{86400000} \quad (6-2)$$

(5) 动态速率：通过改变 timeSpeedKnob 实现-1000x~+1000x 速率调节。

6.3.2 轨迹可视化

轨迹可视化依赖于 Spacekit.js 实现，基本逻辑为前端选择需要加载的卫星，后端计算得到符合 Spacekit.js 要求的轨道参数，并响应，前端利用得到的数据进行卫星加载，该功能时序图如图 6-25 所示。



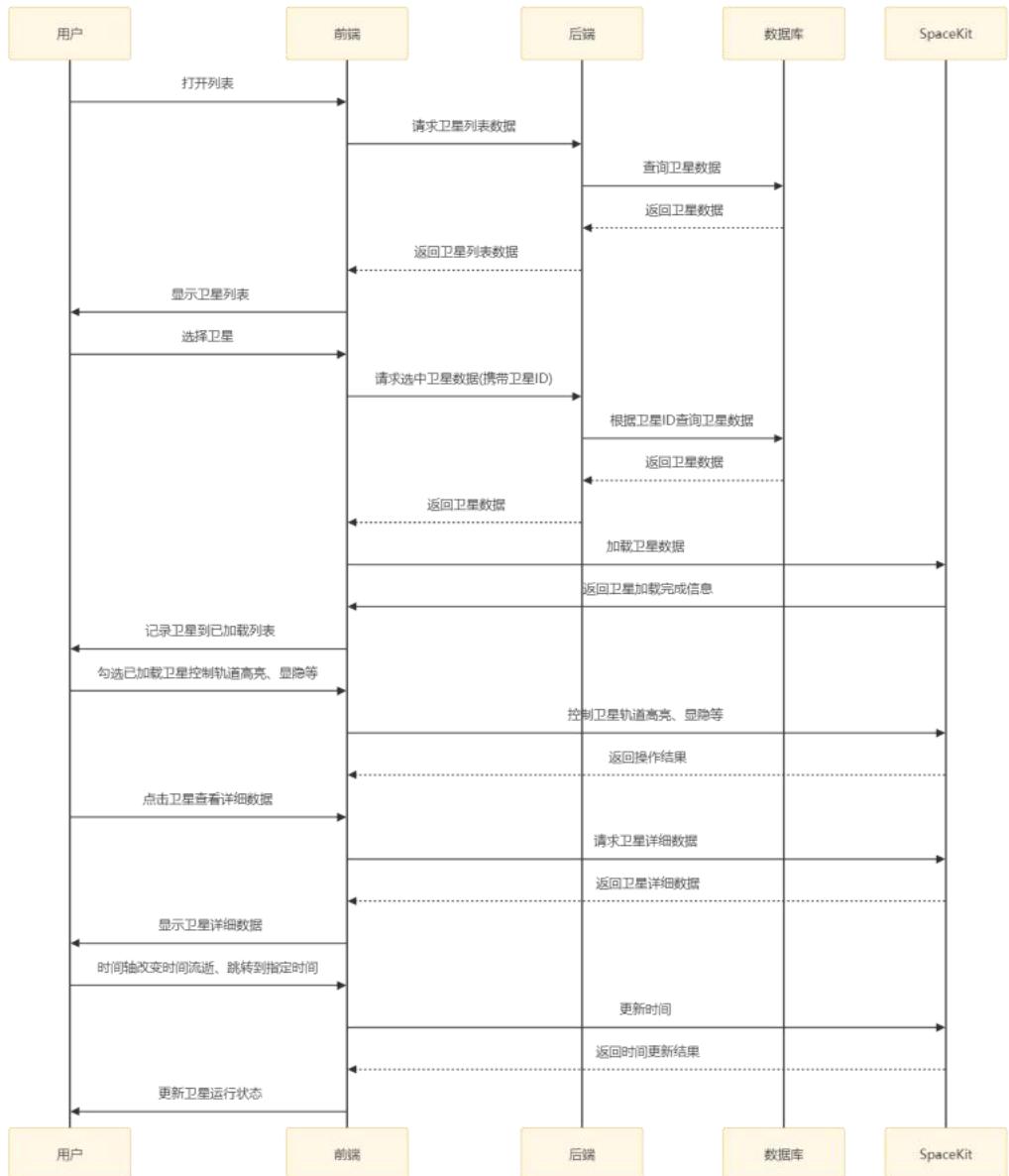


图 6-25 轨迹可视化时序图

关键技术点在于后端轨道参数计算，通过解析 TLE (Two-Line Element Set) 数据，计算并返回卫星的轨道六元素。具体包括：

半长轴 (a)：轨道椭圆的半长轴，单位为公里 (km)。

偏心率 (e)：轨道椭圆的偏心率，无量纲。

轨道倾角 (i)：轨道平面与地球赤道平面的夹角，单位为度。

升交点赤经 (Ω)：轨道平面与地球赤道平面交点的赤经，单位为度。

近地点角 (ω)：从升交点到近地点的角度，单位为度。

平近点角 (M)：从近地点到卫星位置的平角，单位为度。

实现步骤为：

(1) 定义常量 GM。

地球标准重力参数，单位为 km^3/s^2 。其值为 398600.4418。用于计算半长轴。

(2) 创建卫星对象。

通过使用 `twoline2rv` 函数创建卫星对象，基于 SGP4 模型解析 TLE 数据，实现轨道参数的提取。（注：wgs72 是预定义地球引力模型，用于描述地球引力场）

```
satellite = twoline2rv(line1, line2, wgs72)
```

图 6-26 创建卫星对象

(3) 计算半长轴。

方法 1：直接从卫星对象中获取半长轴 a ，单位为地球半径（6371.393 km）

方法 2：通过开普勒第三定律计算半长轴。首先计算平均运动周期（秒），然后转换为弧度/秒，最后通过开普勒第三定律计算半长轴。

两种方法相比较，方法 1 依赖于 TLE 数据的准确性。如果 TLE 数据有误，计算结果也会受到影响。无法验证半长轴的合理性，只能假设 TLE 数据是准确的。因此采用方法 2，具体实现如图 6-27。

```
n = satellite.no_kozai # 平均运动圈/天  
T = 1 / n # 平均运动周期(秒)  
n_rad_s = n * 2 * math.pi / 86400 # 圈/天 → 弧度/秒  
a_km = (GM / (n_rad_s ** 2)) ** (1/3) # 开普勒第三定律计算半长轴(公里)
```

图 6-27 计算半长轴

(4) 偏心率：从卫星对象中直接获取偏心率。

```
e = satellite.ecco # 偏心率
```

图 6-28 获取偏心率

(5) 轨道倾角：从卫星对象中获取轨道倾角 i ，单位为度。

(6) 升交点赤经：从卫星对象中获取升交点赤经 om ，单位为度。

(7) 近地点角：从卫星对象中获取近地点角 w ，单位为度。

(8) 平近点角：从卫星对象中获取平近点角 m ，单位为度。

(9) 历元时间：从卫星对象中获取历元时间 $epoch$ ，单位为儒略日。

由于 Spacekit 中长度单位使用天文单位（以 A.U. 表示，其数值取地球和太

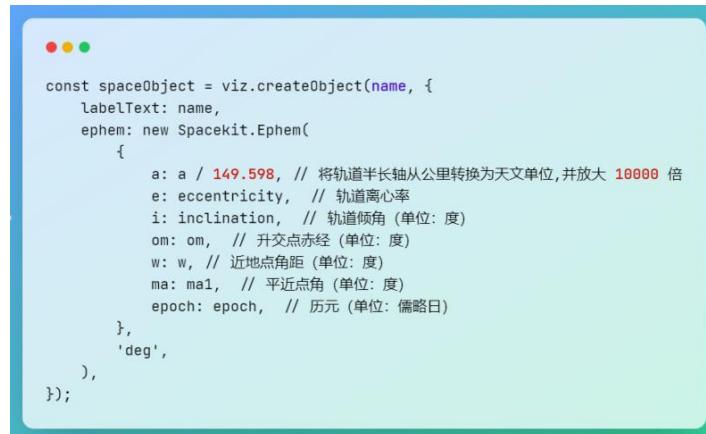
阳之间的平均距离。国际天文学联合会 1964 年决定采用 1A.U.= 1.496×10^8 千米)，因此使用 Spacekit 进行卫星轨迹可视化，需要对涉及长度单位的参数（半长轴 a）进行换算，由于 Spacekit 主要用于天文系统可视化，以太阳为中心，本系统中以地球为中心，因此将地球半径和卫星半长轴合理放大相同倍数，可以在不改变相对位置的同时，获得更好的视觉效果。

因此本系统中，统一将地球半径和卫星半长轴放大 10000 倍，相关代码如图 6-29、图 6-30 所示。



```
var earth = viz.createSphere('earth', {
    textureUrl: 'assets/img/eso_earth.jpg', // 地球纹理图片路径
    // 不透明度
    radius: 6371 / 149.598, // 地球的真实半径，单位为km, 6371 km是地球半径
    // unitsPerAu: 20.0, // 调整大小的单位
    levelsOfDetail: [ // 设置细节层次
        { radii: 0, segments: 64 },
        { radii: 30, segments: 16 },
        { radii: 60, segments: 8 },
    ],
    rotation: { // 旋转设置
        enable: enable, // 允许旋转
        speed: speed, // 自转速度
    };
});
```

图 6-29 生成地球



```
const spaceObject = viz.createObject(name, {
    labelText: name,
    ephem: new Spacekit.Ephem(
        {
            a: a / 149.598, // 将轨道半长轴从公里转换为天文单位，并放大 10000 倍
            e: eccentricity, // 轨道离心率
            i: inclination, // 轨道倾角(单位: 度)
            om: om, // 升交点赤经(单位: 度)
            w: w, // 近地点角距(单位: 度)
            ma: ma1, // 平近点角(单位: 度)
            epoch: epoch, // 历元(单位: 儒略日)
        },
        'deg',
    ),
});
```

图 6-30 生成卫星

6.3.3 过境分析

过境分析主要实现流程为前端获取卫星数据，设置分析时间段和绘制分析区域，通过 satellite.js 模块，根据 TLE (Two-Line Element Set) 数据和分析时间段来生成卫星的轨迹，然后将这个轨迹提供给 Cesium 进行加载，从而在三维场景中模拟卫星的运行效果。同时，使用 Turf.js 这个地理空间分析库来判断卫星是否过境，即卫星的轨迹是否与用户绘制或选择的分析区域有交集，并据此绘制过境线。此外，用户可以通过改变时间轴来查看不同时间点卫星的位置和过境情况。当用户点击轨迹线时，还可以查看卫星的过境时间和出境时间，这些信息会显示

在自定义的信息框中。时序图如图 6-31 所示。

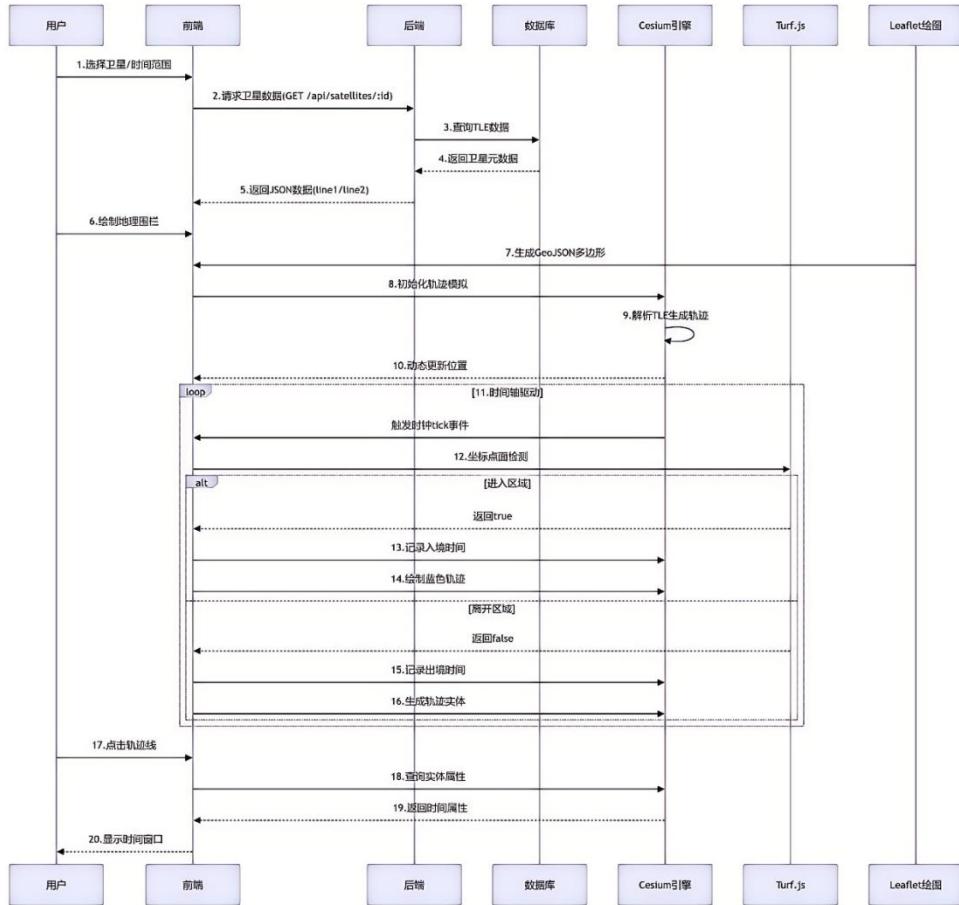


图 6-31 过境分析时序图

关键代码如下：

(1) TLE 数据解析与轨迹生成，主要为通过 Satellite.js 生成卫星的轨迹。代码逻辑如下。

① 构造函数：

初始化卫星对象。使用 twoline2satrec 函数将 TLE 数据转换为 satrec 对象。
从 options 参数中获取配置参数。

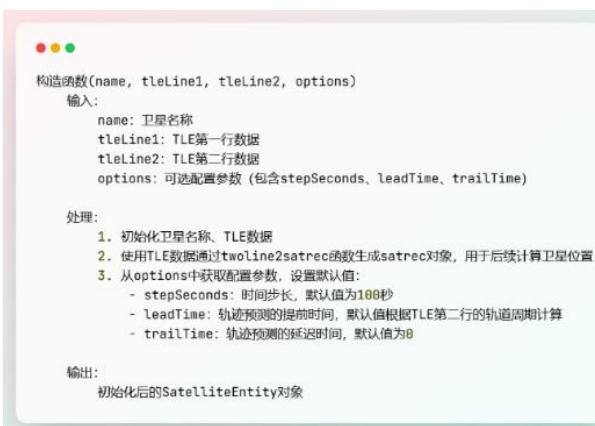


图 6-32 构造函数伪代码

② 获取地心惯性坐标系坐标:

使用 propagate 函数和 satrec 对象以及时间参数计算卫星在地心惯性坐标系中的位置。

satrec 对象是 Satellite.js 库中用于表示卫星轨道的内部数据结构，通过 twoline2satrec 函数从 TLE 数据生成。

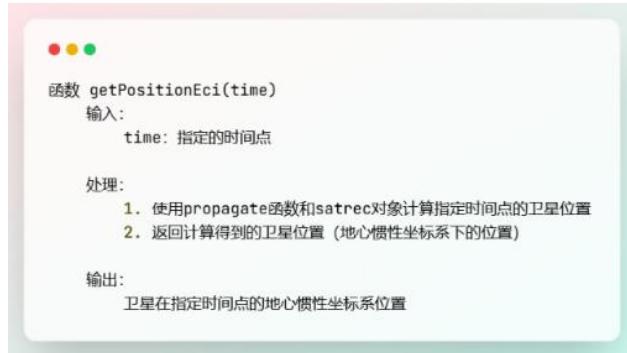


图 6-33 获得地心惯性坐标系

③ `getPositionProperty` 函数:

创建 Cesium.SampledPositionProperty 对象，用于存储卫星在不同时间点的位置数据。计算时间区间内的总秒数和步数。然后遍历时间区间，计算每个时间点卫星的位置，将其添加到 `SampledPositionProperty` 对象中。

`Cesium.SampledPositionProperty` 是一个用于表示采样位置的属性，可以存储卫星在不同时间点的位置信息，并在 Cesium 中动态显示卫星轨迹。



图 6-34 `getPositionProperty` 函数伪代码

④ 创建卫星实体:

转换时间格式，将开始时间和结束时间转换为 `Cesium.JulianDate` 对象。调用 `_getPositionProperty` 方法生成动态路径。构造卫星实体的配置信息，包括名称、描述、时间区间、位置属性、点样式、路径样式和标签样式。



```

● ● ●

函数 createSatelliteEntity(startTime, stopTime)
输入:
    startTime: 开始时间
    stopTime: 结束时间

处理:
    1. 将输入的时间转换为Cesium的JulianDate格式
    2. 调用_getPositionProperty函数生成卫星的位置数据
    3. 创建卫星实体的配置信息, 包括:
        - 名称 (name)
        - 描述 (description)
        - 时间区间 (availability)
        - 位置属性 (position)
        - 点样式 (point)
        - 轨迹样式 (path)
        - 标签样式 (label)
    4. 返回卫星实体的配置信息

输出:
    卫星实体的配置信息

```

图 6-35 创建卫星实体伪代码

⑤ 辅助函数 computePosition

该函数的作用是将输入的经纬度和高度数据转换为 Cesium 所需的 Cartesian3 格式，并存储在 SampledPositionProperty 对象中。

```

● ● ●

函数 computePosition(source, panduan)
输入:
    source: 包含时间、经纬度、高度等信息的数组
    panduan: 判断参数 (用于选择高度类型)

处理:
    1. 初始化一个Cesium的SampledPositionProperty对象
    2. 遍历source数组, 将每个时间点的位置数据转换为Cesium的Cartesian3格式,
    并添加到SampledPositionProperty对象中
    3. 返回SampledPositionProperty对象

输出:
    包含位置数据的SampledPositionProperty对象

```

图 6-36 computePosition 函数伪代码

基于以上逻辑, 得到了可供 Cesium 加载的卫星实体和卫星轨道实体, Cesium 加载卫星和轨道后, 接下来是判断其是否入境。

(2) 入境判断

入境判断的实现思路为 Satellite.js 处理得到卫星实体和轨道实体后, Cesium 进行加载, 然后使用时间轴组件驱动, 实时获取卫星星下点, 通过 Turf.js 判断星下点是否位于划定的分析区域中。

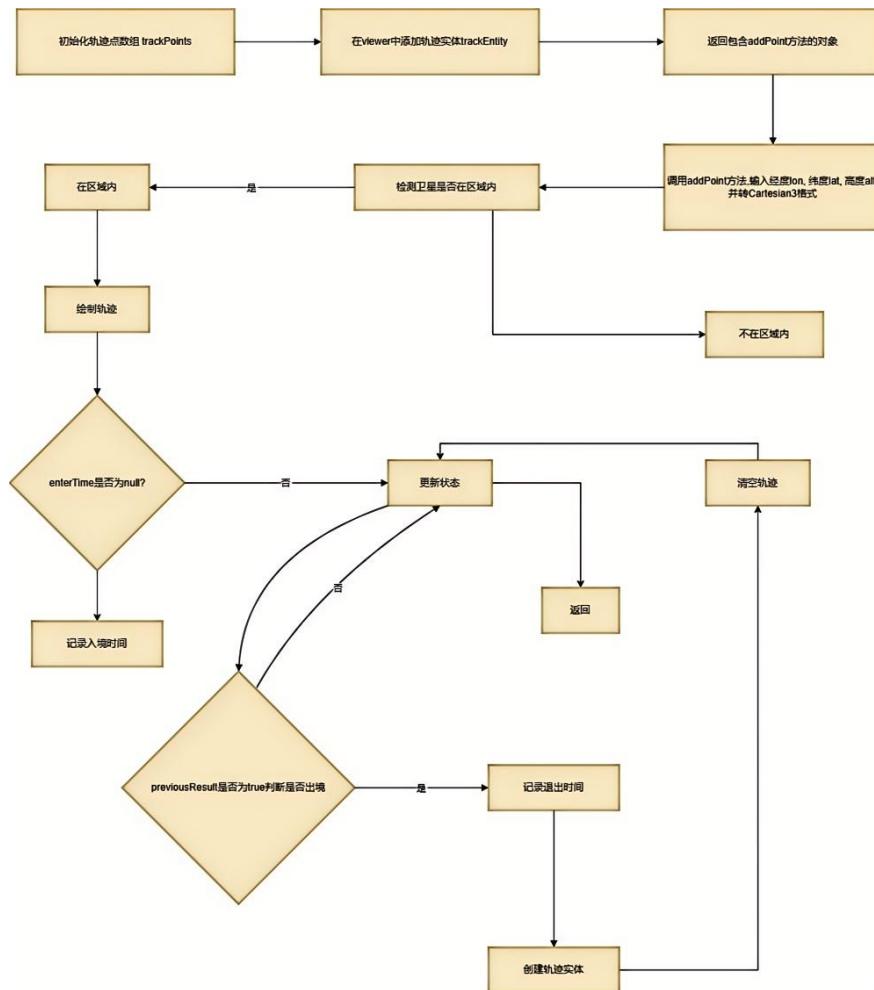


图 6-37 computePosition 函数

① 初始化所需环境和变量

这部分代码的主要目的是初始化轨迹绘制和过境分析所需的环境和变量。通过添加卫星实体到 Cesium 场景，并初始化轨迹绘制功能，为后续的过境分析做好准备。

```

● ● ●
初始化轨迹绘制和过境分析
输入:
  cesiumStore.viewer: Cesium Viewer实例
  cesiumStateEntity: 卫星实体
  publicStore.geojsonPolygons: 区域多边形的GeoJSON数据
  publicStore.satelliteName: 卫星名称

处理:
  1. 添加卫星实体到Cesium Viewer
  2. 初始化过境分析标志位transitAnalysisStart为false
  3. 调用initSingleTrack函数初始化单条轨迹绘制功能
  4. 定义getTime函数, 用于在过境分析开始时获取卫星位置并绘制轨迹

输出:
  初始化完成, 等待过境分析开始

```

图 6-38 初始话相关变量

② 轨迹绘制函数

该函数实现可动态绘制轨迹功能。通过使用 Cesium 的 CallbackProperty，能够实时更新轨迹点，从而实现动态轨迹绘制的效果。并返回 addPoint 方法实现动态追加新的点位到轨迹中。



图 6-39 轨迹绘制函数伪代码

③ 追加点位到轨迹

该函数的核心是通过 Turf.js 库检测点是否在区域内，并根据检测结果决定是否绘制轨迹或记录时间。通过这种方式，可以实现对卫星轨迹是否过境的实时分析，并在 Cesium 中动态显示轨迹。

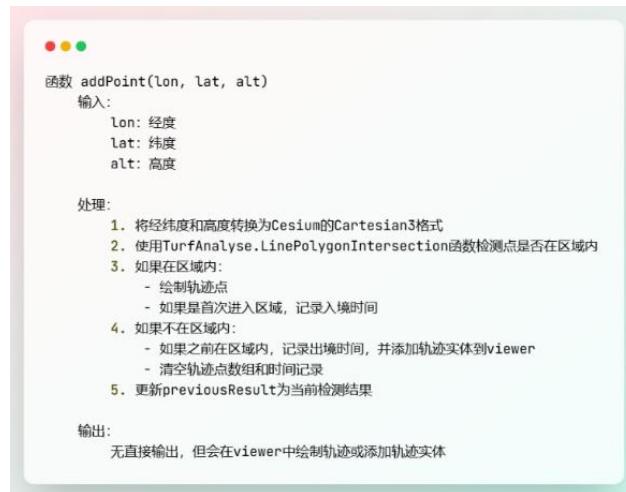


图 6-40 轨迹点追加函数伪代码

④ 入境分析

该类的目的是封装 Turf.js 的空间分析功能。通过提供点是否在多边形内方法（LinePolygonIntersection），实现对卫星轨迹是否过境的检测，返回结果。

```

● ● ●

类 TurfAnalyse
    静态方法 LinePolygonIntersection(point, polygon)
        输入:
            point: 点坐标
            polygon: 多边形的GeoJSON数据

        处理:
            1. 将点坐标转换为Turf.js的点要素
            2. 从GeoJSON中提取多边形要素
            3. 使用Turf.js的booleanPointInPolygon函数检测点是否在多边形内
            4. 返回检测结果

        输出:
            提供空间分析功能

```

图 6-41 入境分析函数伪代码

6.3.4 卫星模拟

卫星模拟功能的目的是通过让用户输入轨道参数，实现对卫星轨道的模拟。

用户通过输入长半轴 (a)、偏心率 (e)、轨道倾角 (i) 通过 `createOrbit` 函数生成卫星的轨道信息，并通过 `sateDot` 函数将卫星和轨道添加到 Cesium 场景中，实现卫星轨道的模拟显示。

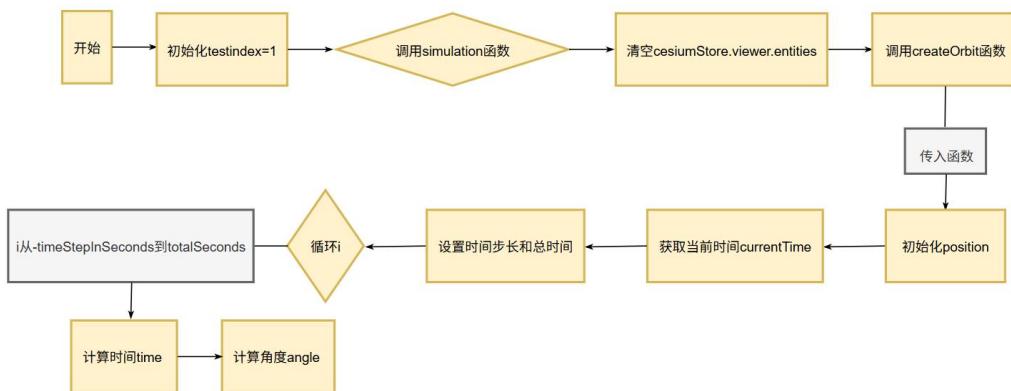


图 6-42 卫星模拟逻辑流程图

核心代码如下：

(1) simulation 函数

清空 Cesium 场景中的所有实体，以便重新绘制。调用 `createOrbit` 函数，根据传入的参数（高度、偏心率、倾角）生成轨道数据，然后将 `sateDot` 函数返回的卫星点和轨道信息添加到场景中。`testindex` 自增 1，用于区分不同的卫星。

```

● ● ●
初始化变量 testIndex = 1

函数 simulation():
    清空 cesiumStore.viewer.entities 中的所有实体
    调用 createOrbit 函数, 传入 heightValueArr.value[0],
    heightValueArr.value[1]、heightValueArr.value[2],
    取得轨道信息
    调用 cesiumStore.viewer.entities.add 函数, 传入
    stateDot 函数的返回值 (卫星点和轨道信息)
    testIndex 自增 1

```

图 6-43 simulation 函数伪代码

(2) createOrbit 函数

初始化 position 为 Cesium.SampledPositionProperty 对象, 用于存储卫星在不同时间的位置。

获取当前时间 currentTime, 设置时间步长 timeStepInSeconds 和总时间 totalSeconds。

使用 for 循环, 从 -timeStepInSeconds 到 totalSeconds, 步长为 timeStepInSeconds:

- ① 计算当前时间点 time。
- ② 根据椭圆轨道的参数 (长半轴 majorAxis、偏心率 eccentricity) 和当前角度 angle, 计算卫星的位置坐标(x,y,z)。
- ③ 计算绕 X 轴旋转的旋转矩阵 rotation, 并将旋转矩阵应用到位置向量 positionVector 上, 模拟卫星轨道的倾角。
- ④ 将当前时间 time 和计算得到的位置向量 positionVector 添加到 position 中。
- ⑤ 返回包含 position 的对象, 用于表示卫星的轨道信息。

```

● ● ●
函数 createOrbit(majorAxis, eccentricity, inclination):
    初始化 position 为 Cesium.SampledPositionProperty 对象
    获取当前时间 currentTime
    设置时间步长 timeStepInSeconds = 600 (秒)
    设置总时间 totalSeconds = 86400 (秒)

    对于 i 从 -timeStepInSeconds 到 totalSeconds, 步长为
    timeStepInSeconds:
        计算时间 time = currentTime + i 秒
        计算角度 angle = (i / totalSeconds) * 2π
        计算 x = majorAxis * 1000 * cos(angle)
        计算 y = majorAxis * 1000 * sin(angle) * sqrt(1 -
        eccentricity ** 2)
        设置 z = 110
        计算绕 X 轴旋转的旋转矩阵 rotation
        初始化位置向量 positionVector = (x, y, z)
        将旋转矩阵 rotation 应用到 positionVector 上
        将 time 和 positionVector 添加到 position 中

    返回 { position }

```

图 6-44 createOrbit 函数

(3) stateDot 函数

stateDot 函数实现返回一个对象，包含卫星的标识信息 (id)、标签 (label)、位置 (position)、点样式 (point) 和路径样式 (path)。

通过设置 leadTime (轨道预测长度) 和 trailTime (轨道回溯长度) 为较大的值，使得轨道始终完整展示，而不是随时间动态绘制。

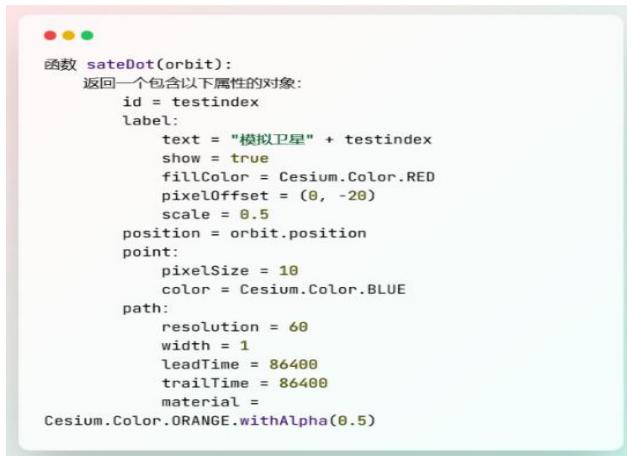


图 6-45 stateDot 函数

轨道六参数中与轨道位置相关的参数为长半轴 a 、偏心率 e 、轨道倾角 i 、升交点赤经 Ω 、近地点幅角 ω ；与时间相关的参数为平近点角 M 。考虑到用户对轨道六参数中部分参数了解较少，因此对模拟参数进行默认假设和简化处理，使得其他轨道参数（如升交点赤经 Ω 、近地点幅角 ω 和平近点角 M ）被省略或默认设置为特定值。使得用户只需要输入长半轴 (a)、偏心率 (e)、轨道倾角 (i) 即可快速模拟轨道。

针对升交点赤经 Ω 和近地点幅角 ω 的模拟，轨道的初始位置是通过角度 θ （从 0 到 2π ）来计算的，而没有显式地使用升交点赤经 Ω 和近地点幅角 ω 。因此：

① 升交点赤经 Ω 默认为 0，假设轨道平面的升交点与参考方向（通常是赤道平面的正方向）重合。

② 近地点幅角 ω 默认设置为 0，假设近地点位于升交点处。

使得轨道的初始位置从近地点开始，默认从近地点开始绘制轨道。

针对平近点角 M 的模拟，是通过让卫星的位置通过时间步长 Δt 和总时间 T 来计算的，而不是直接使用平近点角 M 。具体来说通过如下方式实现：

① 时间步长 Δt 和总时间 T 用于生成轨道上的多个点：通过时间步长 Δt 从 0 到 T 生成多个时间点，每个时间点对应一个卫星位置。

② 轨道周期隐含：总时间 T 足够长，可以覆盖一个完整的轨道周期，使得角度 θ 从 0 到 2π 的变化隐含了平近点角 M 的变化

③ 近地点起始位置假设：假设初始时刻卫星位于近地点，即平近点角 $M=0$ ，

随着时间的推移，角度 θ 逐渐增加，隐含了平近点角 M 的变化。

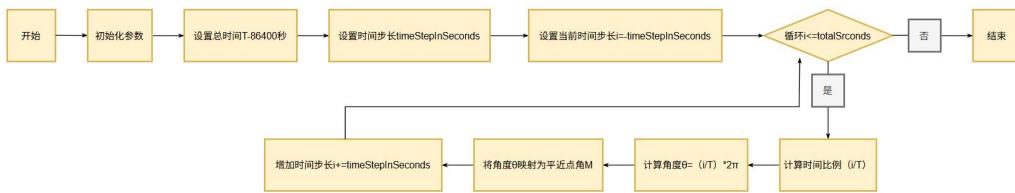


图 6-46 平近点角模拟逻辑

对平近点角 M 的具体简化处理逻辑如下：

① 平近点角 M 假设

平近点角 M 是从近地点开始，卫星在轨道上扫过的角度，它是时间的函数。

对于一个椭圆轨道，平近点角 M 与时间 t 的关系可以表示为：

$$M(t) = n(t - t_0) \quad (6-3)$$

其中 n 是卫星的平均角速度，与轨道周期 T 和长半轴 a 有关。 t_0 是卫星通过近地点的时刻。

② 角度θ的计算：

通过时间比例计算角度θ：

$$\theta = \frac{i}{T} \times 2\pi \quad (6-4)$$

其中 i 是时间步长（从 $-timeStepInSeconds$ 到 $totalSeconds$ ）。T 是总时间（86400 秒，即一天）。

③ 隐含平近点角 M 的变化：

时间步长 i 从 $-timeStepInSeconds$ 到 $totalSeconds$ 逐步增加。每个时间步长 i 对应一个角度θ，这个角度θ是平近点角 M 的一个近似表示。角度θ从 0 到 2π 的变化，隐含平近点角 M 从 0 到 2π 的变化，以此实现角度θ随时间的变化隐含平近点角 M 的变化。

④ 轨道周期的隐含：

总时间 T 设置为 86400 秒，隐含一个假设的轨道周期。实际的轨道周期 T_{orbit} 与长半轴 a 有关，本系统通过假设总时间 T 足够长，以此覆盖一个完整的轨道周期。

⑤ 近地点的起始位置：

假设初始时刻 ($i=0$) 卫星位于近地点，这相当于平近点角 $M=0$ 。

基于以上逻辑，随着时间的推移，角度 θ 逐渐增加，这隐含了平近点角 M 随时间的增加。

6.3.5 轨迹预测

系统获取用户选择的卫星 ID。和选择的时间范围，调用后端服务获取卫星的 TLE 数据。使用 TLE 数据和时间范围，利用 Satellite.js 计算得到卫星轨迹，然后将卫星实体加载到 Cesium 场景中，完成轨迹分析。

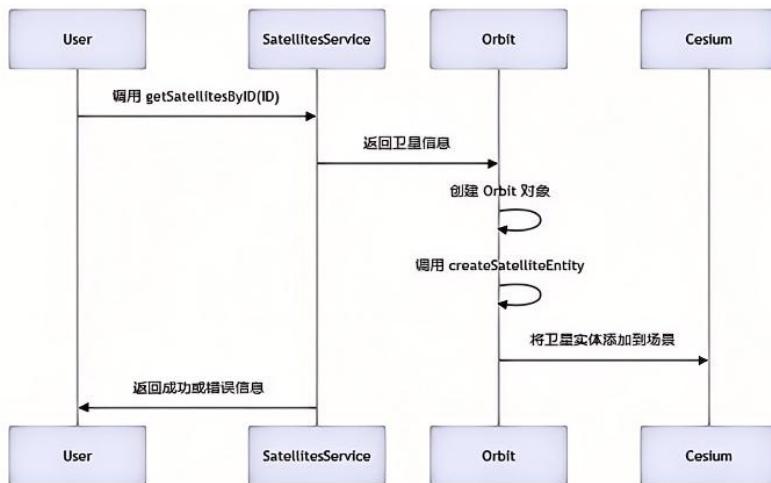


图 6-47 轨迹分析时序图

核心逻辑如下：

- (1) 创建 Orbit 对象：使用获取到的卫星名称、TLE 数据和时间范围，创建 Orbit 对象。Orbit 对象负责解析 TLE 数据，并计算卫星在指定时间段内的轨迹。



图 6-48 创建 Orbit 对象

- (2) Orbit 对象使用 Satellite.js 库中的 twoline2satrec 和 propagate 方法，解析 TLE 数据并计算卫星在每个时间点的位置。将计算得到的位置存储到 Cesium.SampledPositionProperty 中，以便在 Cesium 中动态显示。

```

● ● ●

类 Orbit:
构造函数(name, tleLine1, tleLine2, timeField, options):
    初始化卫星名称、TLE 数据
    解析 TLE 数据为 satrec 对象
    设置总时间、时间步长、预报周期和延迟时间

方法 _checkTle(tle):
    检查 TLE 数据格式是否正确
    如果格式错误，抛出异常
    返回解析后的 TLE 数据

方法 getPositionEci(time):
    使用 satrec 对象和给定时间，计算卫星的地心惯性坐标系位置
    返回位置坐标

方法 _getPositionProperty(start, startTime1):
    初始化一个 SampledPositionProperty 对象
    获取当前时间
    将字符串时间转换为时间戳
    循环生成卫星在不同时间的位置：
        计算当前时间点
        获取卫星位置
        如果位置有效，将时间点和位置添加到 SampledPositionProperty 中
    返回 SampledPositionProperty 对象

方法 createSatelliteEntity(startTime, stopTime):
    将开始时间和结束时间转换为 Cesium 的 JulianDate 格式
    设置卫星实体的属性，包括名称、描述、可用性、位置、点样式、路径样式和标签
    返回卫星实体

```

图 6-49 Orbit 类

6.3.6 通信分析

地面站与卫星之间的通信分析，主要步骤为设置通信距离阈值和时间范围。创建地面站静态实体，调用后端服务获取卫星的 TLE 数据，并生成卫星轨迹。根据地面站与卫星之间的距离，动态显示或隐藏连接线（通信链路）。

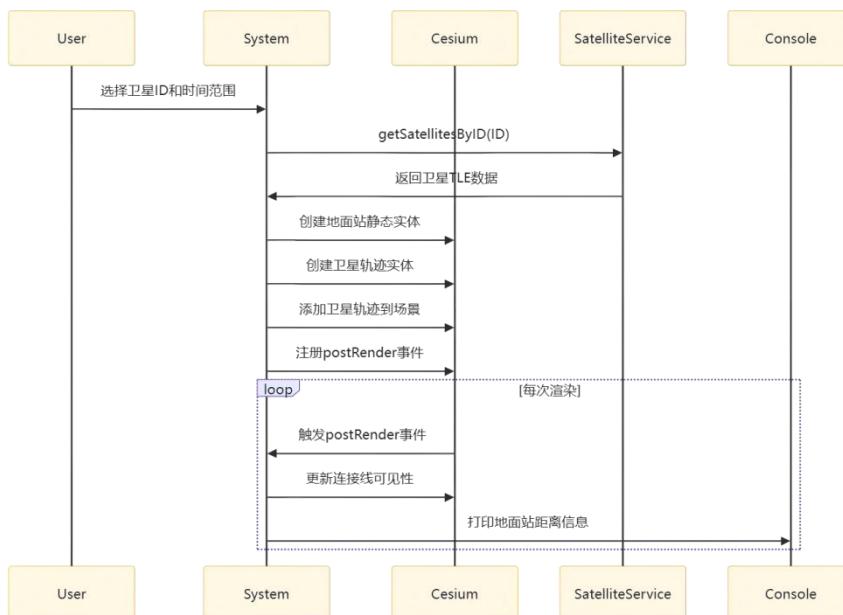


图 6-50 通信分析时序图

具体实现逻辑如下：

(1) 初始化变量

distanceNum：设置地面站与卫星之间的通信距离阈值，单位为千米。

`groundAnalysisdates`: 存储用户选择的时间范围，用于分析卫星与地面站的通信情况。

(2) 创建地面站静态实体

定义了三个地面站的位置、名称和颜色。使用 `Cesium.Cartesian3.fromDegrees` 将经纬度转换为三维坐标。为每个地面站创建一个 Cesium 实体，包含位置、点样式和标签，并将这些实体添加到 Cesium 场景中。

(3) 获取卫星数据

从全局存储中获取用户选择的卫星 ID。调用 `SatellitesService.getSatellitesByID(ID)` 获取卫星的 TLE 数据。通过 `Satellite.js` 创建 `SatelliteEntity` 对象，并生成卫星轨迹实体添加到 Cesium 场景中。



```
// 初始化变量
distanceNum = 5000 // 距离阈值，单位为千米
groundAnalysisdates = [] // 时间范围数组

函数 groundStationAnalysis():
    // 创建地面站静态实体
    staticEntities = [
        { position: (16.644356, 16.986161, 100), name: "地面站1", color: BLUE },
        { position: (-75.59777, 48.83883, 100), name: "地面站2", color: GREEN },
        { position: (139.6917, 35.6895, 100), name: "地面站3", color: ORANGE }
    ]
    对于每个 config 在 staticEntities:
        创建一个 Cesium 实体，包含位置、点样式和标签
        将实体添加到 Cesium 场景中

    // 获取用户选择的卫星ID
    ID = publicStore.DataListSelectSatelliteID

    // 调用后端服务获取卫星数据
    调用 SatellitesService.getSatellitesByID(ID)
    如果成功:
        创建 SatelliteEntity 对象
        创建卫星轨迹实体
        将卫星轨迹实体添加到 Cesium 场景中
        调用 tick 函数，传入卫星实体、地面站实体和距离阈值
    如果失败:
        打印错误信息
```

图 6-51 地面站分析事件函数伪代码

(4) 动态更新连接线的可见性

在 `tick` 函数中，为每个地面站创建一个连接线，连接地面站和卫星。使用 `Cesium.CallbackProperty` 动态更新连接线的位置。注册 Cesium 场景的 `postRender` 事件，每次渲染时动态更新连接线的可见性。计算地面站和卫星间的距离，距离小于阈值，显示连接线，否则隐藏连接线。

```

● ● ●

函数 tick(dynamicEntity, staticEntities, distanceNum):
    // 为每个地面站创建连接线
    connectionLines = []
    对于每个 entity 在 staticEntities:
        创建一个 Cesium 多段线实体，连接地面站和卫星
        将多段线实体添加到 connectionLines

    // 动态更新所有连线的可见性
    注册场景的 postRender 事件
    在事件回调中:
        获取当前时间
        获取卫星的当前位置
        对于每个地面站:
            获取地面站的当前位置
            计算地面站和卫星之间的距离
            如果距离小于阈值，显示连接线，否则隐藏连接线
            打印地面站的距离信息

```

图 6-52 动态更新连接线显隐函数伪代码

6.3.7 覆盖分析

覆盖分析通过用户指定周期，从后端计算指定卫星周期内指定卫星的星下点并返回，前端获取数据后，Cesium 加载星下点、连点成线，得到覆盖轨迹。

后端依赖 skyfield 库计算卫星星下点，得到周期内卫星星下点坐标，响应前端请求。

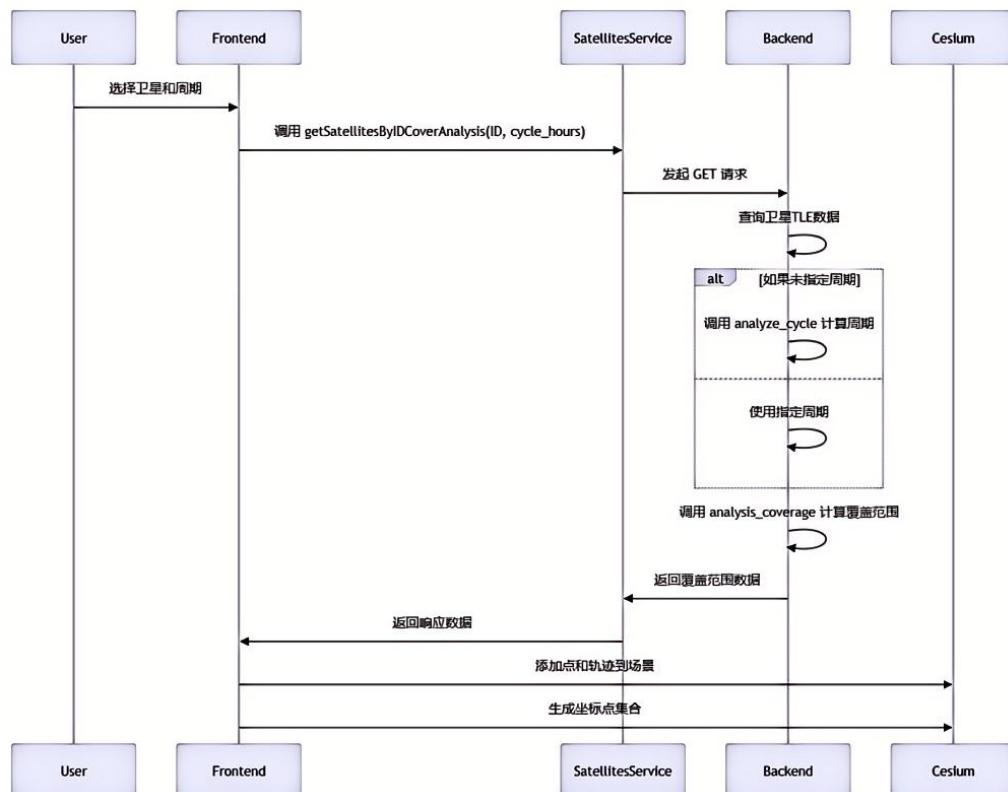


图 6-53 覆盖分析时序图

具体实现逻辑如下：

(1) 数据请求

前端调用 `SatellitesService.getSatellitesByIDCoverAnalysis`, 传入卫星 ID 和周期。卫星服务通过 Axios 向后端服务发起 GET 请求, 获取卫星覆盖分析数据。

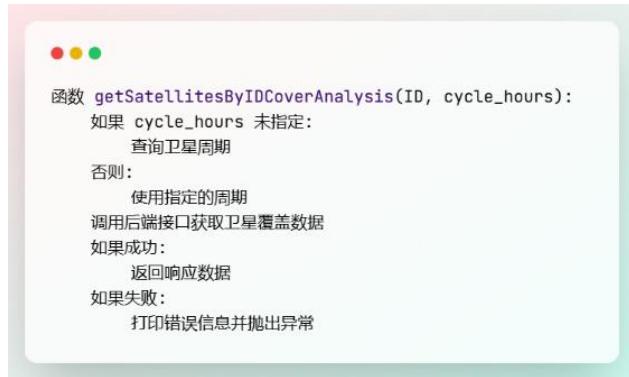


图 6-54 请求数据

(2) 后端服务处理请求

后端服务接收请求, 查询卫星的 TLE 数据。如果未指定周期, 调用 `analyze_cycle` 计算卫星周期。

调用 `analysis_coverage` 计算卫星覆盖范围。得到计算结果后, 计算得到的星下点数据返回给前端。

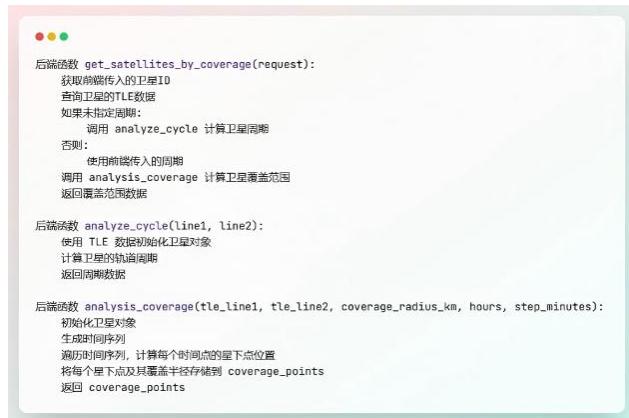


图 6-55 计算卫星覆盖范围

(3) 展示覆盖情况

卫星服务将响应数据返回给前端。前端遍历响应数据, 通过得到的星下点经纬度坐标添加星下点实体到 Cesium 场景中。调用 `CesiumEntity.addPolyline` 根据星下点数据, 连点成线, 展示覆盖情况。



图 6-56 展示覆盖情况

6.3.8 数据统计

(1) 数据总览

数据总览逻辑为前端调用后端数据统计接口，获取各类统计结果进行展示。数据总览时序图如图 5-54 所示，具体实现功能如下：

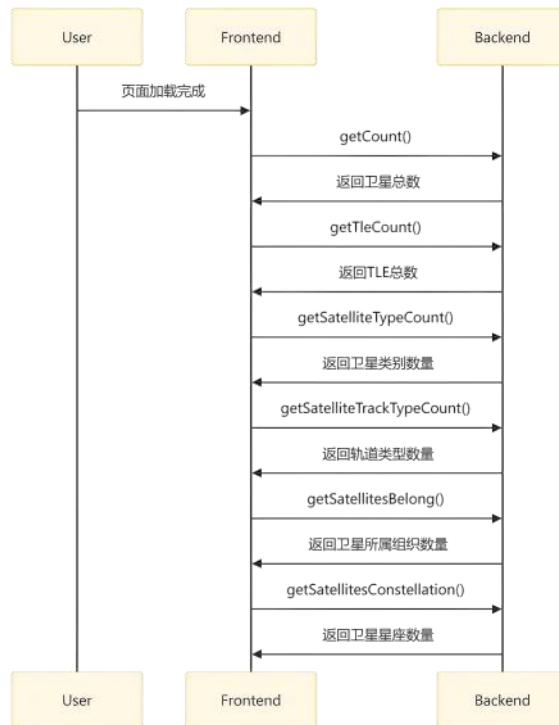


图 6-57 数据总览时序图

① 获取卫星总数

前端调用 `SatellitesService.getCount()`，请求后端获取卫星总数。后端返回卫星总数，前端将进行展示。

② 获取 TLE 总数

前端调用 `SatellitesService.getTleCount()`，请求后端获取 TLE 总数。后端返回 TLE 总数，前端将进行展示。

③ 获取卫星类别数量

前端调用 SatellitesService.getSatelliteTypeCount(), 请求后端获取卫星类别数量和卫星类别, 前端将结果进行展示。

④ 获取轨道类型数量

前端调用 SatellitesService.getSatelliteTrackTypeCount(), 请求后端获取轨道类型数量和轨道类型, 前端将结果进行展示。

⑤ 获取卫星所属组织数量

前端调用 SatellitesService.getSatellitesBelong(), 请求后端获取卫星所属组织数量和卫星所属组织类型, 前端将结果进行展示。

⑥ 获取卫星星座数量

前端调用 SatellitesService.getSatellitesConstellation(), 请求后端获取卫星星座数量和卫星星座类型, 前端将结果进行展示。

(2) 卫星所属分类

通过前端调用 getSatellitesBelong 方法, 请求后端获取卫星所属国家(组织或地区)的统计信息。后端处理请求, 查询每个国家的卫星数量将查询结果转换为列表。返回 JSON 响应。前端接收并处理响应数据后将数据存储到状态变量中。使用 ECharts 初始化图表实现展示。

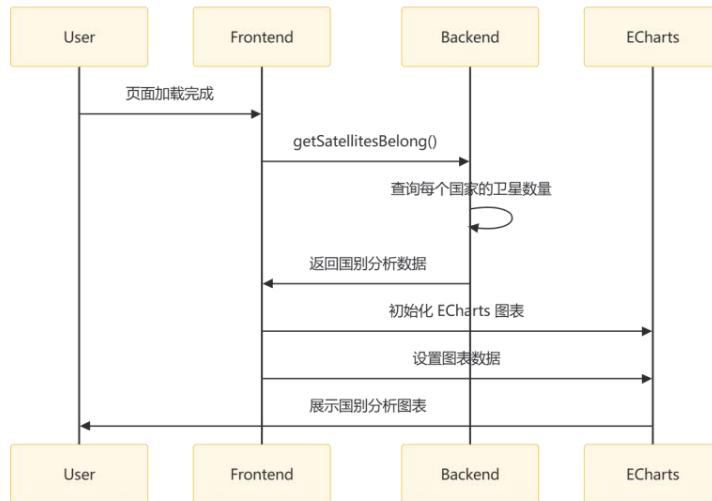


图 6-58 卫星所属分类时序图

(3) 卫星轨道类型及卫星所属类型分类

前端调用后端接口前端调用 SatellitesService.getTypeCombinationNumber(), 请求后端获取卫星类型组合数量。调用 SatellitesService.getSatelliteTypeCount(), 请求后端获取卫星类型数量。调用 SatellitesService.getTracktTypeCombinationNumber(), 请求后端获取轨道类型组合数量。调用 SatellitesService.getSatelliteTrackTypeCount(), 请求后端获取轨道类型数量。

后端接收到请求后，调用相应的视图函数：

- ① `get_satellite_combination_number_by_type`: 查询卫星类型组合数量
- ② `get_satellite_type_count`: 查询卫星类型数量。
- ③ `get_satellite_type_count`: 查询卫星类型数量。
- ④ `get_satellite_combination_number_by_tracktype`: 查询轨道类型组合数量。
- ⑤ `get_satellite_tracktype_count`: 查询轨道类型数量。

使用 Django ORM 进行查询，并将结果转换为列表格式。返回 JSON 响应。

前端接收到后端返回的数据。将数据存储到状态变量中。使用 ECharts 初始化图表，并设置图表数据。

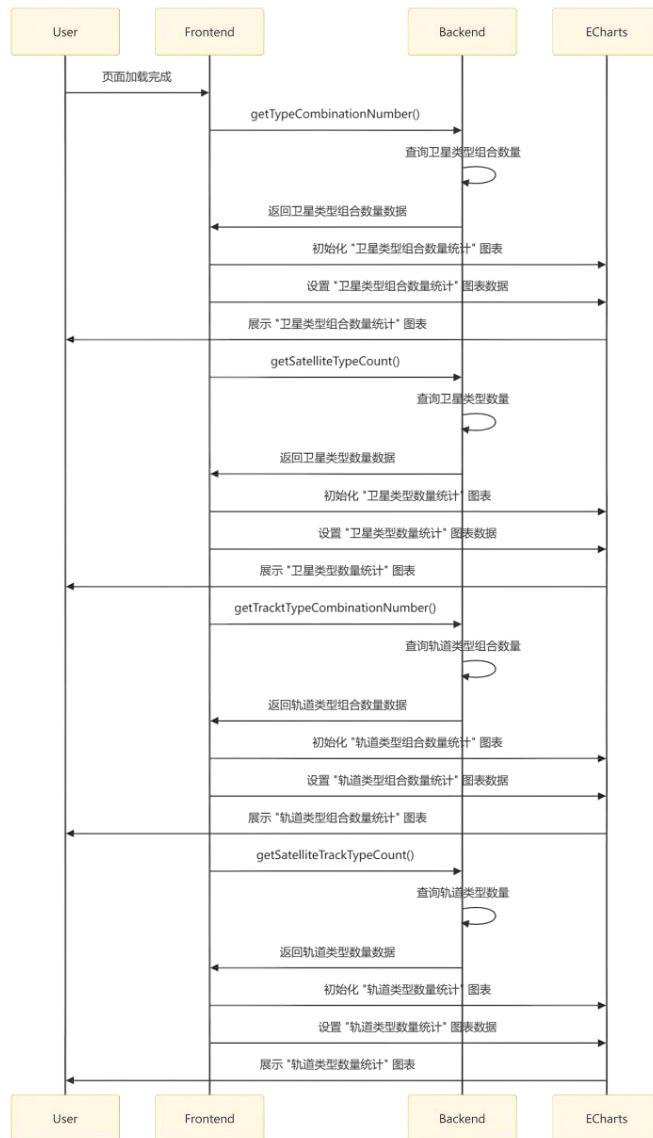


图 6-59 卫星轨道类型及卫星所属类型分类时序图

(4) 星座分类

星座分类逻辑为前端发送请求，后端统计数据库中卫星星座种类，并统计各

类星座的卫星数量，然后响应数据，前端根据数据绘制柱状图展示。

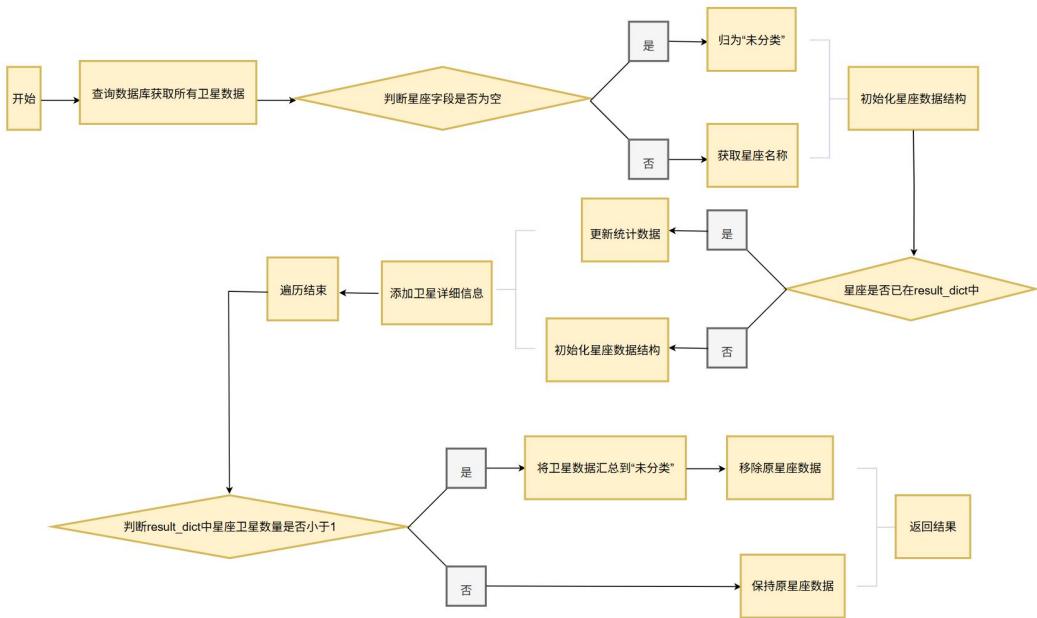


图 6-60 星座分类流程图

核心实现逻辑为：

查询数据库，获取所有卫星的星座、ID、名称和所属字段，初始化一个空字典，用于存储按星座分类的卫星统计结果。遍历查询结果，按星座分类。

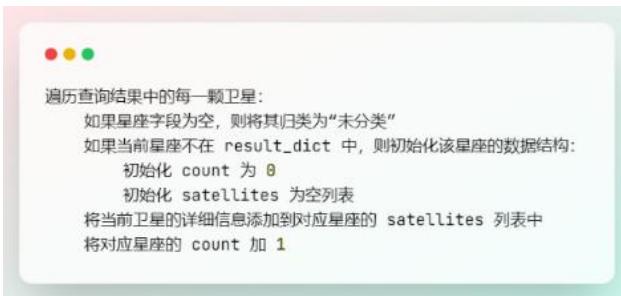


图 6-61 数据查询

处理卫星数量小于 1 的星座（无意义，单颗卫星不构成星座）。

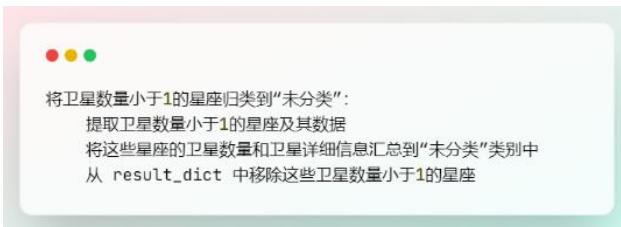


图 6-62 数据处理

6.3.9 数据分析

(1) 轨道高度分析

实现逻辑为前端调用后端接口前端调用

SatellitesService.getHeightWithTimeById(id)，请求后端获取卫星高度随时间变化的数据。调用 SatellitesService.getSatellitePositionByCycle(id)，请求后端获取卫星周期内位置变化的数据。

后端接收到请求后，调用相应的视图函数。get_height_with_time_by_id 获取卫星高度随时间变化的数据。getSatellitePositionByCycle 获取卫星周期内位置变化的数据。将结果转换为列表格式。最后返回 JSON 响应。

前端接收到后端返回的数据后，将数据存储到状态变量中，使用 ECharts 初始图表，并设置图表数据。

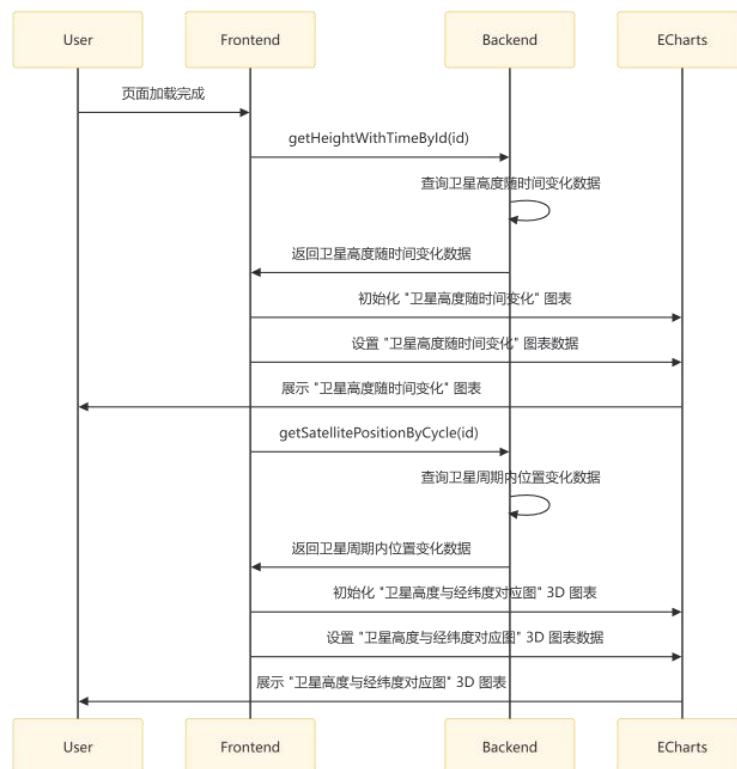


图 6-63 轨道高度分析时序图

核心代码逻辑为计算卫星最低点高度、最高点高度和平均高度，返回 JSON 格式数据，包含卫星最低点高度、最高点高度和平均高度数据。

```
● ● ●
函数 get_height_extremes_and_average(request):
    获取前端传入的卫星ID
    根据卫星ID查询卫星的TLE数据
    调用 analyze_cycle 函数计算卫星周期
    如果返回的周期是字符串，尝试解析为JSON格式
    获取周期内的小时数（默认24小时）
    调用 analyze_height_extremes_and_average 函数计算高度极值和平均值
    将结果解析为JSON格式并返回
```

图 6-64 高度极值与平均高度接口入口函数

分析卫星周期内的高度极值和平均值，以及波动范围和极值对应时刻和位置，输入参数为 TLE 数据分析时长（小时）、时间步长（分钟）。

```
●●●  
函数 analyze_height_extremes_and_average(line1, line2, hours=24, step_min=0.5):  
    使用 TLE 数据初始化卫星对象  
    计算卫星的平均运动值和周期  
    将周期转换为小时数  
    初始化天体数据  
    生成时间序列  
    遍历时间序列，计算每个时间点的卫星高度  
    构建元数据，包括卫星信息和轨道参数  
    返回 JSON 格式的元数据
```

图 6-65 高度极值与平均高度计算

计算卫星周期内位置变化并返回 JSON 数据，输入参数：卫星 ID、计算时长（单位：小时）、计算步长（单位：分钟），返回：JSON 格式数据，包含卫星周期内位置变化数据。

```
●●●  
函数 get_locations_by_hours(request):  
    获取前端传入的卫星ID  
    根据卫星ID查询卫星的TLE数据  
    如果未指定计算时长，调用 analyze_cycle 函数计算卫星周期  
    如果返回的周期是字符串，尝试解析为JSON格式  
    获取周期内的小时数（默认24小时）  
    如果未指定计算步长，默认为1分钟  
    调用 analyze_locations_with_cycle 函数计算周期内卫星位置变化  
    将结果返回
```

图 6-66 卫星周期内位置变化接口入口函数

依赖 skyfield 库 EarthSatellite 方法，使用 TLE 数据，根据时间序列，计算每个时间点卫星的位置，得到卫星高度随时间、随经纬度变化情况的数据。

```
●●●  
函数 analyze_locations_with_cycle(tle_line1, tle_line2, hours, step_minutes=1):  
    使用 TLE 数据初始化卫星对象  
    初始化天体数据  
    生成时间序列  
    遍历时间序列，计算每个时间点的卫星位置（纬度、经度、高度）  
    将每个时间点的位置信息记录到列表中  
    返回位置信息列表
```

图 6-67 计算卫星周期内位置变化

(2) 卫星速度分析

速度分析功能为实现展示卫星速度周期内随时间变化情况。实现逻辑为前端调用 SatellitesService.getSpeedWithTimeById(id)，请求后端获取卫星速度随时间变化的数据。

后端接收到请求后，调用 get_speed_with_time_by_id 视图函数。使用 Django ORM 查询卫星的 TLE 数据。调用 analyze_speed 函数计算卫星速度。将结果以 JSON 格式返回。

前端接收到后端返回的数据后，将数据存储到状态变量中。使用 ECharts 初始化图表，并设置图表数据。

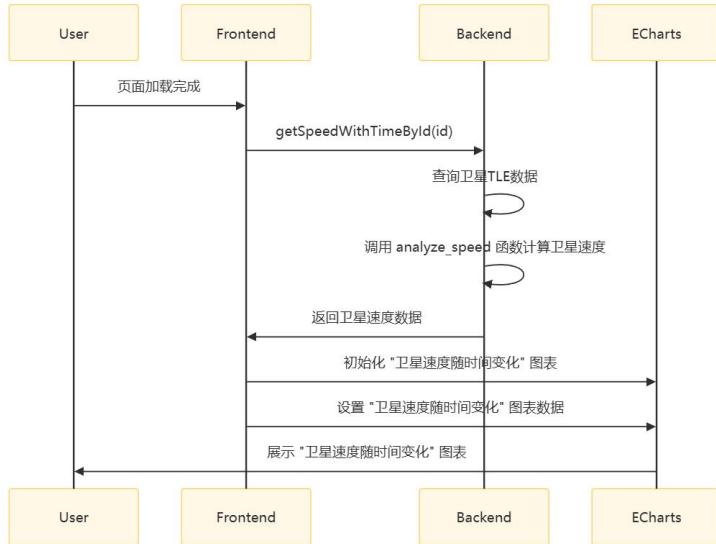


图 6-68 卫星速度分析时序图

核心代码逻辑为依赖 skyfield 库 EarthSatellite 方法，使用 TLE 数据，根据时间序列，计算每个时间点卫星的位置和速度，得到卫星速度随时间变化情况的数据。

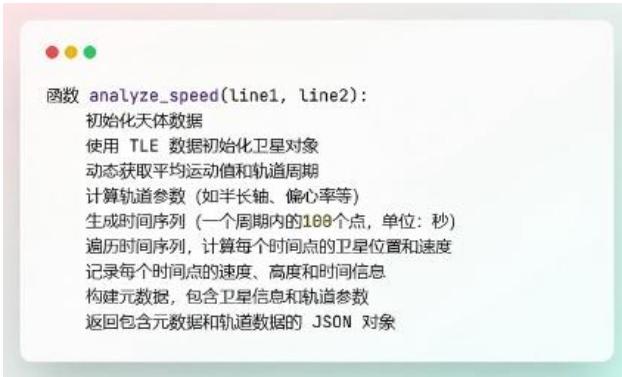


图 6-69 计算卫星速度随时间变化

(3) 星下点分析

星下点实现效果为获取当前时刻所有卫星星下点，使用热力图展示其分布情况，实现思路为前端调用 SatellitesService.getStarsDownPointNow()，请求后端获取当前时间所有卫星的位置（星下点）。后端接收到请求后，调用 get_satellite_location_now 视图函数。查询所有卫星的 TLE 数据。遍历每个卫星，计算其当前星下点位置。将所有成功计算的卫星星下点位置以 JSON 格式返回。

前端接收到后端返回的卫星位置数据。使用 Leaflet 初始化地图。使用 L.heatLayer 添加热力图图层，显示星下点的热力分布。

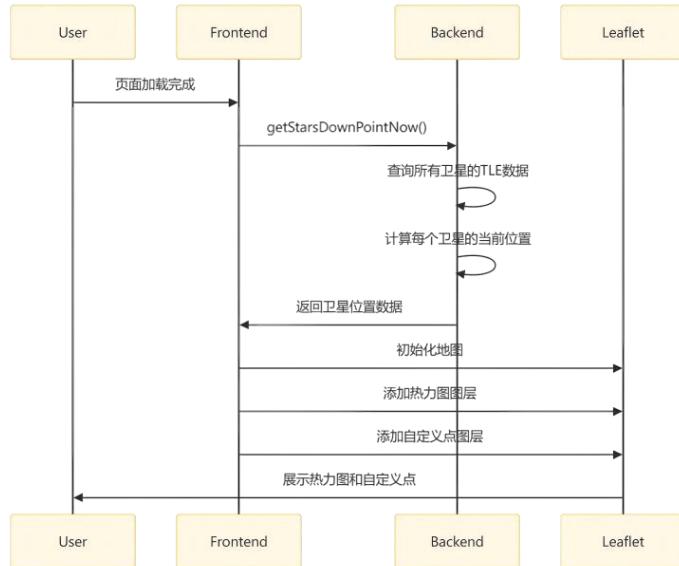


图 6-70 星下点分析时序图

核心逻辑（图 6-71）为根据 TLE 数据和 UTC 时间，计算卫星星下点位置。

```

● ● ●
函数 calculate_subpoint(tle_line1, tle_line2, dt_utc):
    # 解析TLE数据
    satellite = Satrec.twoline2rv(tle_line1, tle_line2)

    # 将datetime对象分解为时间组件
    year = dt_utc.year
    month = dt_utc.month
    day = dt_utc.day
    hour = dt_utc.hour
    minute = dt_utc.minute
    second = dt_utc.second + dt_utc.microsecond / 1e6

    # 计算儒略日和时间部分
    jd, fr = jday(year, month, day, hour, minute, second)

    # 计算卫星的位置 (ECI坐标系, 单位: 千米)
    error, position, velocity = satellite.sgp4(jd, fr)
    if error ≠ 0:
        抛出异常("SGP4计算错误: {error}")

    x_eci, y_eci, z_eci = position

    # 计算格林尼治平恒星时 (GMST)
    jd_total = jd + fr # 当前时间的儒略日
    d = jd_total - 2451545.0 # 自J2000以来的天数
    gmst_deg = (280.46061837 + 366.98564736629 * d) % 360
    gmst_rad = math.radians(gmst_deg)

    # 将ECI坐标转换为ECEF坐标
    cos_gmst = math.cos(gmst_rad)
    sin_gmst = math.sin(gmst_rad)
    x_ecef = x_eci * cos_gmst + y_eci * sin_gmst
    y_ecef = -x_eci * sin_gmst + y_eci * cos_gmst
    z_ecef = z_eci

    # 计算经度和纬度 (地心坐标) 高度
    earth_radius_km = 6378.137 # 地球赤道半径(km)
    longitude_rad = math.atan2(y_ecef, x_ecef)
    longitude_deg = math.degrees(longitude_rad)
    latitude_rad = math.atan2(z_ecef, math.sqrt(x_ecef**2 + y_ecef**2))
    latitude_deg = math.degrees(latitude_rad)
    height_km = math.sqrt(x_ecef**2 + y_ecef**2 + z_ecef**2) - earth_radius_km

    # 将经度调整到[-180, 180]范围
    longitude_deg = (longitude_deg + 180) % 360 - 180

    返回 (latitude_deg, longitude_deg, height_km)

```

图 6-71 星下点计算伪代码

(4) 卫星详情



卫星详情通过前端请求指定卫星数据（接口 `get_satellites`），通过卫星 TLE 数据调用接口（`get_satellite_elemnets`）获取轨道参数，调用接口（`get_satellite_track_image`）获取卫星轨道图，调用接口（`get_satellite_near_far_height_speed`）获取近/远地点高度和速度，调用接口（`get_height_extremes_and_average`）获取高度极值和平均高度及波动范围数据，调用接口（`get_historytle`）获取该卫星历史 TLE 数据。获取数据后，对其展示。

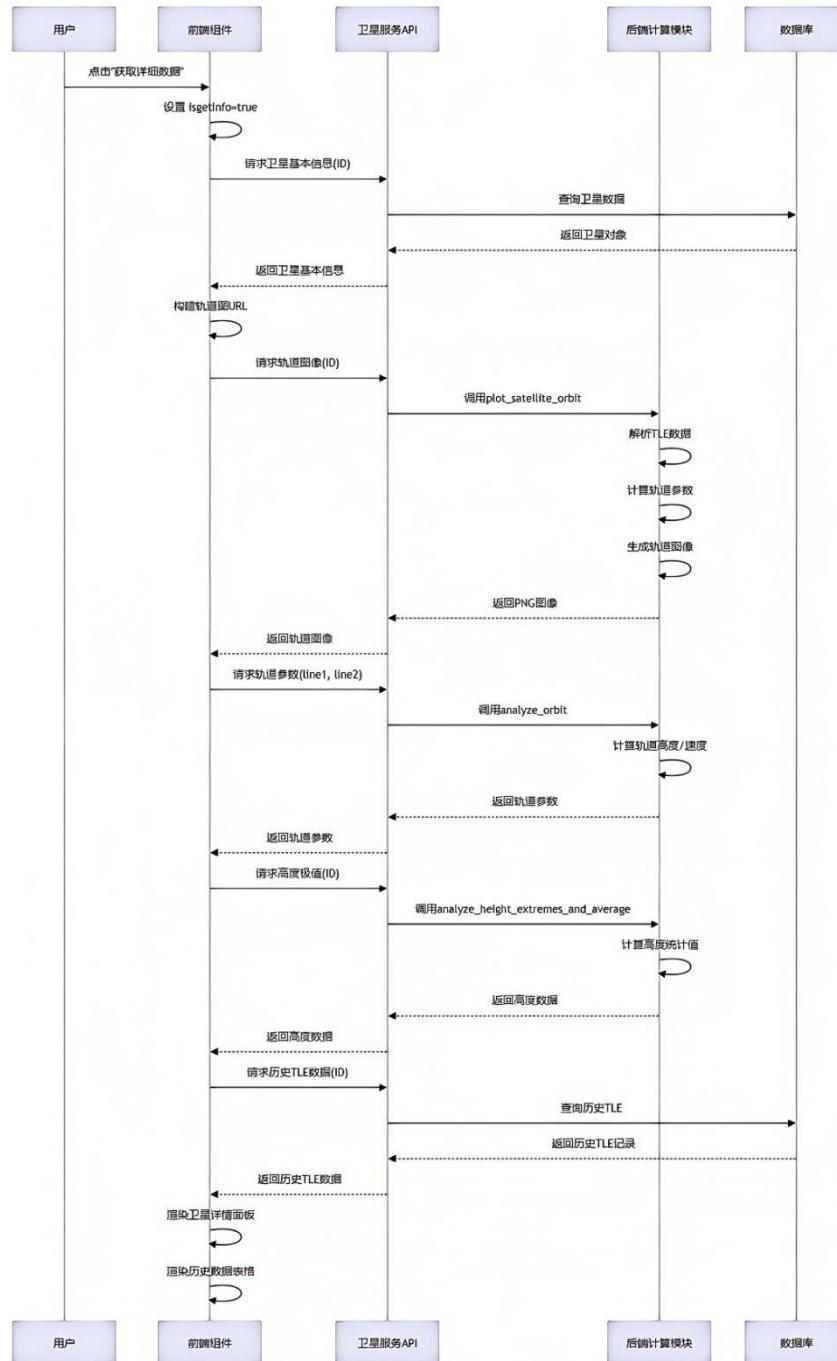


图 6-72 卫星详情展示时序图

功能实现逻辑中，除获取卫星轨道图接口（`get_satellite_track_image`）逻辑和获取该卫星历史 TLE 数据接口（`get_historytle`）逻辑之外，其余功能实现逻辑已在前文提及。获取历史 TLE 数据接口（`get_historytle`）实现逻辑与前文数据查询逻辑相同，因此接下来对卫星轨道图的生成相关逻辑进行展示。

`get_satellite_track_image` 接口通过 `plot_satellite_orbit` 函数实现卫星轨道图的绘制。

```
def plot_satellite_orbit(line1, line2):
    # 解析TLE参数
    # 计算轨道参数
    # 生成轨道坐标
    # 计算近远地点
    # 创建图形
    # 绘制地球和轨道
    # 标注近远地点
    # 设置样式
    # 保存图像
    # 返回图像数据
```

图 6-73 `plot_satellite_orbit` 函数

该函数在对 TLE 进行解析，获取关键轨道参数后，生成轨道图像，主要包括如下步骤：

(1) 生成轨道坐标

① 生成 1000 个点的极坐标角度 θ （从 0 到 2π ）。

② 根据轨道极坐标公式计算半径 r :

$$r = \frac{a(1 - e^2)}{1 + e \cos \theta} \quad (6-5)$$

③ 将极坐标转换为直角坐标系下的 x 和 y 。

```
theta = np.linspace(0, 2 * np.pi, 1000)
r = a * (1 - eccentricity ** 2) / (1 + eccentricity * np.cos(theta))
x, y = r * np.cos(theta), r * np.sin(theta)
```

图 6-74 生成极坐标点

④ 坐标系旋转：将轨道坐标系统绕原点旋转 ω 角度（转换为弧度），以匹配地球坐标系。

```
omega_rad = np.radians(omega)
x_rot = x * np.cos(omega_rad) - y * np.sin(omega_rad)
y_rot = x * np.sin(omega_rad) + y * np.cos(omega_rad)
```

图 6-75 坐标系旋转

(2) 计算近地点和远地点

- ① 近地点距离 r_{peri} 和远地点距离 r_{apo} 公式:

$$r_{\text{peri}} = a(1 - e) \quad \text{和} \quad r_{\text{apo}} = a(1 + e) \quad (6-6)$$

- ② 计算近地点高度 $\text{altitude}_{\text{peri}}$ 和远地点高度 $\text{altitude}_{\text{apo}}$ （相对于地球表面）。

```
earth_radius = 6371 # 地球平均半径, 单位: km
r_peri = a * (1 - eccentricity)
r_apo = a * (1 + eccentricity)
altitude_peri = r_peri - earth_radius
altitude_apo = r_apo - earth_radius
```

图 6-76 计算近地点和远地点高度

- ③ 计算近地点和远地点在旋转后坐标系中的位置。

```
x_peri_rot = r_peri * np.cos(omega_rad)
y_peri_rot = r_peri * np.sin(omega_rad)
x_apo_rot = -r_apo * np.cos(omega_rad)
y_apo_rot = -r_apo * np.sin(omega_rad)
```

图 6-77 旋转后坐标

(3) 绘制轨道图

创建图形后根据上述计算结果和相关常量，绘制地球和轨道、标注近地点和远地点，进行相关样式设置和添加图例，返回图像数据。

6.3.10 数据管理

通过自定义命令解析器实现了以下功能：

- (1) 初始化：配置参数别名、数字字段、数组字段、范围字段和必填字段。
- (2) 解析命令（parse 方法）：将输入命令分解为命令和参数。
- (3) 分词（tokenize 方法）：将输入字符串分解为令牌。
- (4) 解析参数（parseArgs）：将参数解析为键值对。
- (5) 规范化键名（normalizeKey）：将短参数名转换为长参数名。
- (6) 解析值（parseValue）：将值解析为数字或日期。
- (7) 解析范围值（parseRangeValue）：将范围值解析为对象。
- (8) 命令处理：根据命令调用相应的处理方法。
- (9) 验证必填字段：检查必填字段是否已提供。

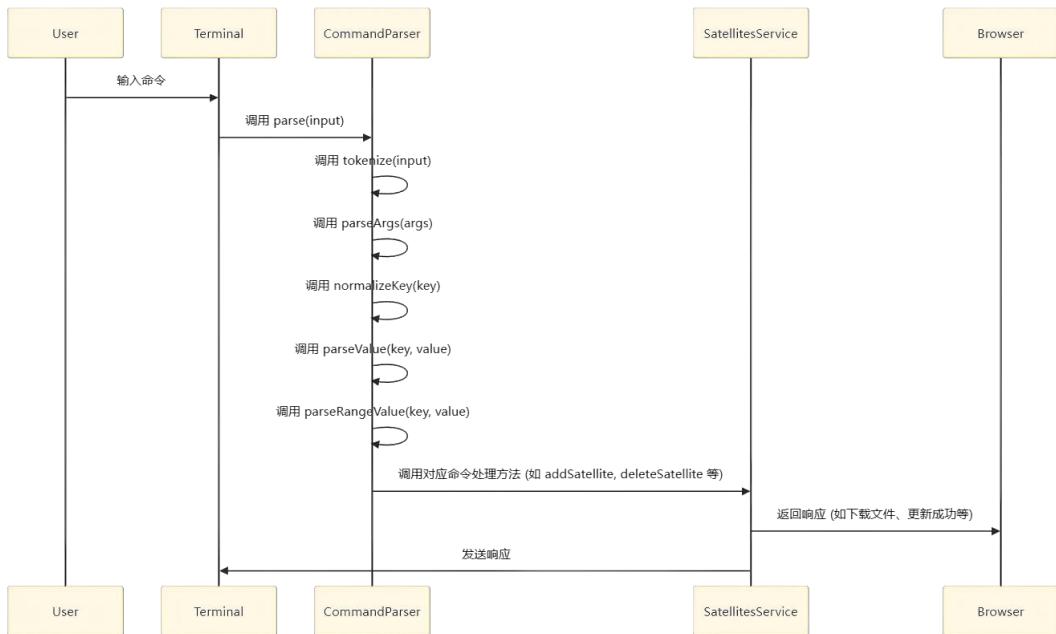


图 6-78 数据管理操作时序图

实现功能详情如下：

(1) 删除记录

通过输入输入 del 命令，提供定位字段，即可删除数据。支持批量删除（使用非唯一字段定位）

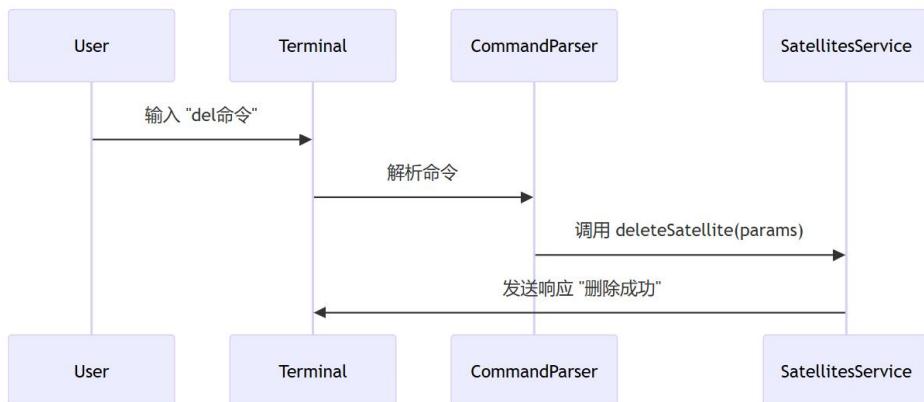


图 6-79 删除操作时序图

del 命令解析方法如下。



图 6-80 删除命令解析方法

del 命令示例：

```
● ● ●  
del --id <卫星ID>      # 按主键删除 (优先级高)  
del --inid <国际编号>    # 按国际卫星标识符删除
```

图 6-81 删命令示例

(2) 更新记录

通过输入输入 update 命令，提供定位字段，后接需要更改的字段和值，即可更新数据。支持批量更新（使用非唯一字段定位）。

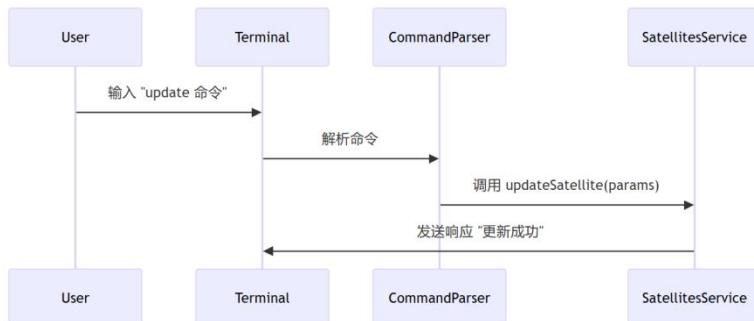


图 6-82 更新操作时序图

update 命令解析方法如下。

```
● ● ●  
函数 handleUpdate(params):  
    调用 SatellitesService.updateSatellite(params)  
    发送响应 "更新成功"
```

图 6-83 更新命令解析方法

使用 id 或者 inid 定位记录。

```
● ● ●  
update --id <卫星ID> [参数]      # 按主键定位  
update --inid <国际编号> [参数]    # 按国际编号定位
```

图 6-84 更新命令定位字段示例

后接需要修改的字段及内容。

```
● ● ●  
update --inid 00001 --year 2024 --country CHN  
update --id 111111 --year 2024 --country CHN
```

图 6-85 更新命令示例

支持批量更新，使用非唯一字段定位，后接需修改字段与内容。



图 6-86 批量操作命令示例

(3) 导出数据

通过输入输入 export 命令，指定下载的数据，即可更新数据。支持按条件下载（通过字段定位）。

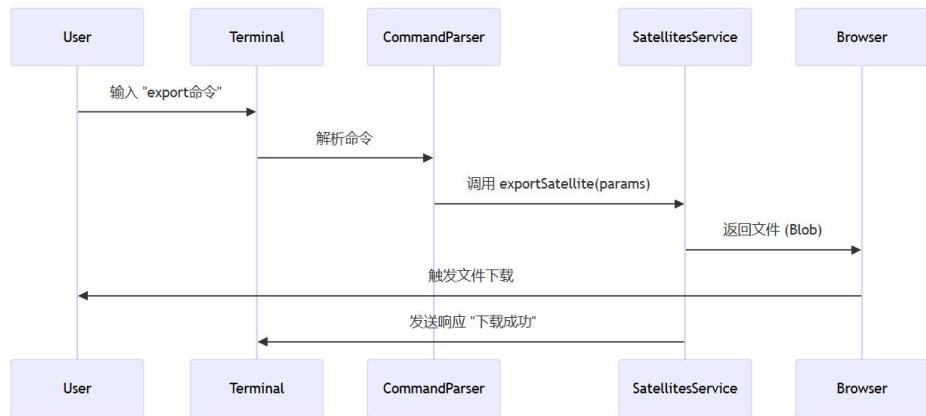


图 6-87 导出数据时序图

export 命令解析方法如下。



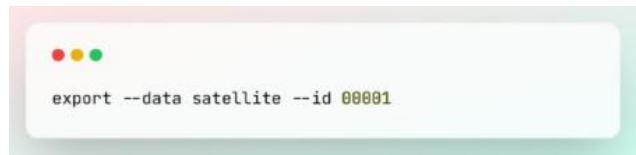
图 6-88 导出数据命令解析方法

使用 export 命令，使用--data 指定导出的数据。



图 6-89 导出数据命令示例

导出特定记录。



```
● ● ●  
export --data satellite --id 00001
```

图 6-90 导出特定记录

按条件导出，使用非唯一字段定位。



```
● ● ●  
export --data satellite --country 'CHN'
```

图 6-91 按条件导出

系统测试与分析

第七章

空天守望者：人造卫星态势感知系统

第七章 系统测试与分析

7.1 功能测试

7.1.1 时间轴

在轨迹可视化和轨迹分析模块，时间轴可对场景时间实现控制。

时间轴组件可显示当前时间，实现时间跳转、时间流逝加减与恢复，时间流速指定，拖动时间轴可控制时间变化。

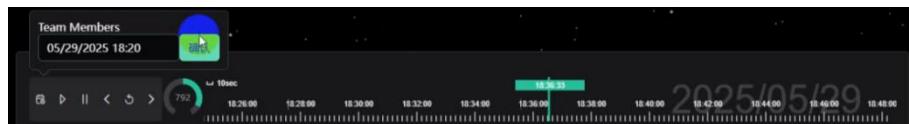


图 7-1 时间轴组件

7.1.2 轨迹可视化

轨迹分析功能实现卫星数据加载，通过待加载列表展示待加载卫星，已加载列表展示已加载到场景中的卫星，选取已加载卫星可以对其轨道、名称标签实现显隐控制，在场景中点击卫星标签可对其高亮显示，并展示卫星信息。

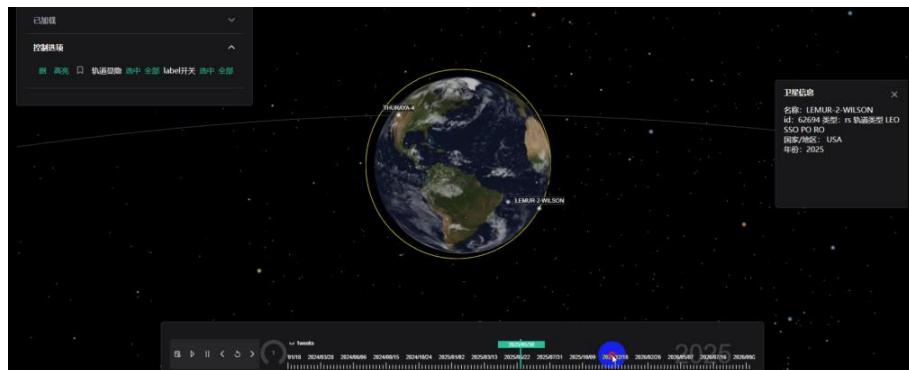


图 7-2 轨迹可视化

7.1.3 轨迹分析

(1) 过境分析

过境分析通过选取卫星、绘制或选取待分析区域、选择分析时间段进行分析。

场景中加载卫星和轨迹后，由时间驱动场景变化，绘制卫星过境线，点击卫星过境轨迹可查看卫星入境时间和出境时间。

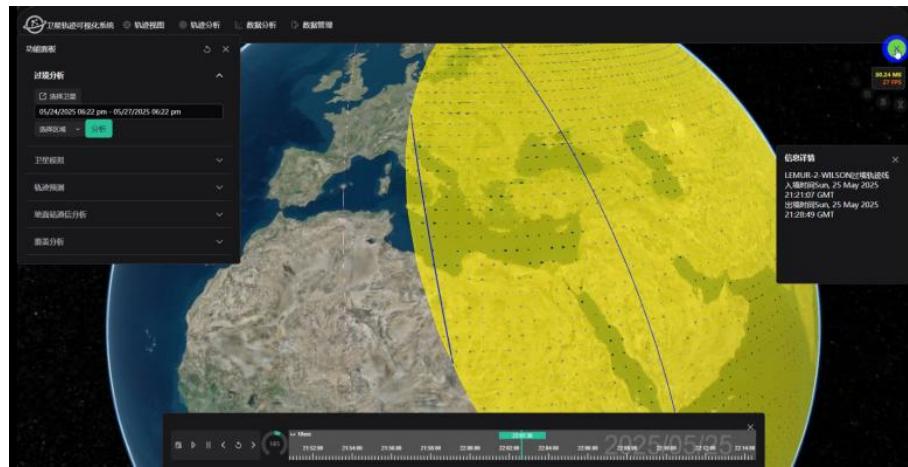


图 7-3 过境分析

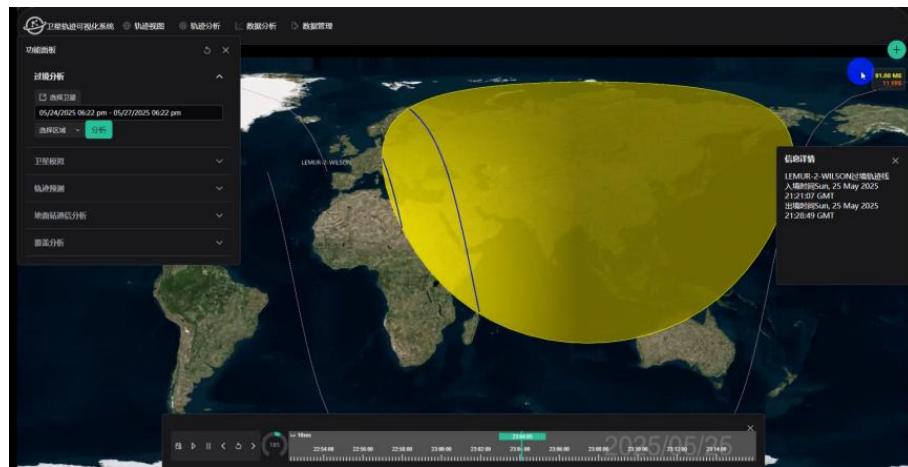


图 7-4 过境分析二维视角

(2) 卫星模拟

卫星模拟通过用户输入卫星长半轴、倾角、偏心率生成卫星和卫星轨迹，实现其运行效果模拟。

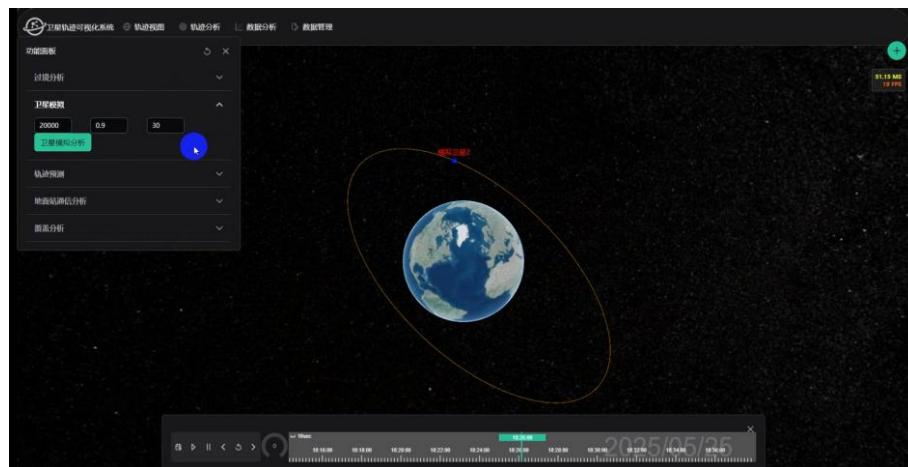


图 7-5 卫星模拟

(3) 轨迹预测

轨迹预测通过用户选择待预测卫星，给定预测时长（如未指定，系统将根据卫星周期进行预测），系统生成卫星轨迹并展示于场景中。

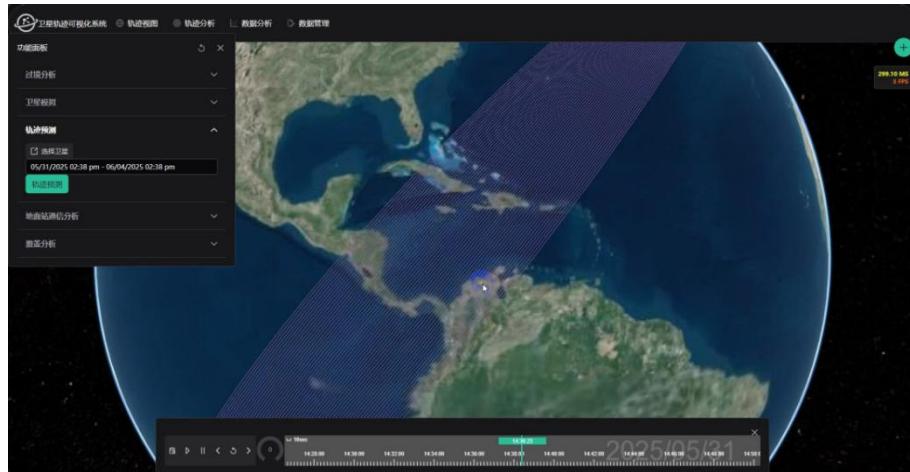


图 7-6 轨迹预测

(4) 通信分析

通信分析通过用户选择待分析卫星，设置可通信距离，场景中加载卫星和卫星轨迹后，由时间驱动场景变化，系统判断当前时刻卫星与地面站距离以控制二者间通信链路的显隐。

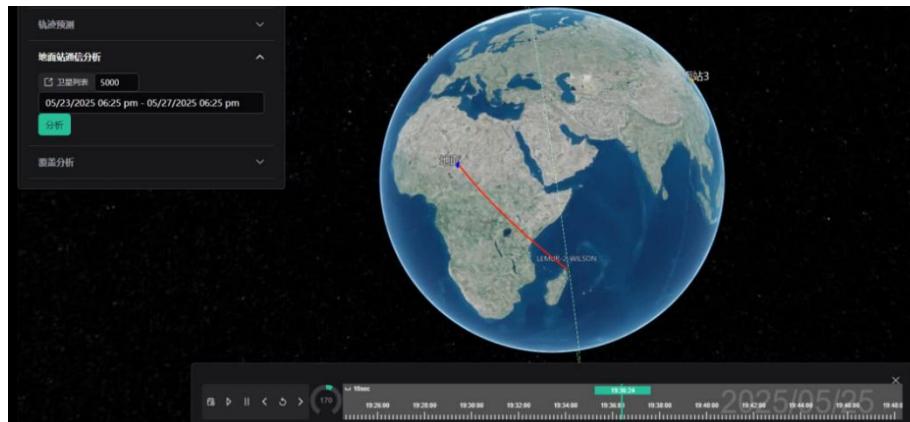


图 7-7 通信分析

(5) 覆盖分析

覆盖分析通过选择卫星、待分析卫星和分析时长，获得分析时段内卫星星下点数据，场景中加载卫星星下点和覆盖轨迹线。



图 7-8 覆盖分析

7.1.4 数据分析

(1) 轨道高度分析

轨道高度分析通过分析用户所选卫星在其周期内，轨道高度随时间和经纬度变化数据，以及观测周期、时间分辨率（分析所用时间步长）、高度极值、波动范围、平均值、高度极值对应位置和时刻。系统对上述数据进行展示。

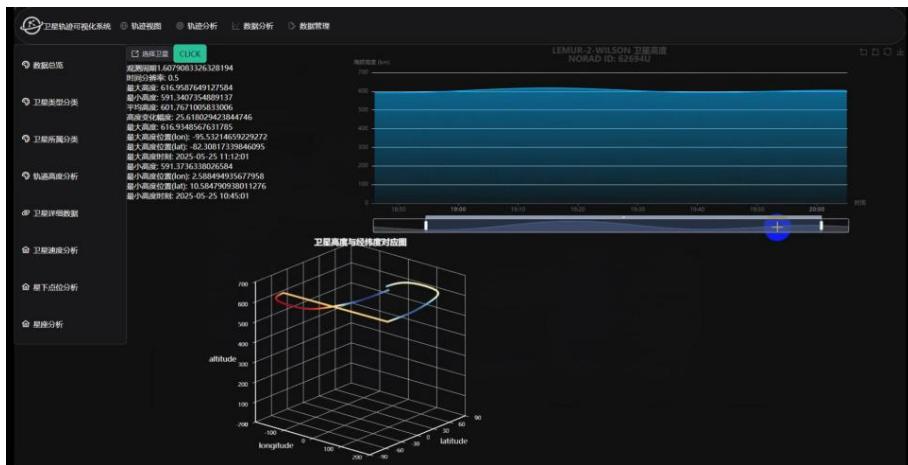


图 7-9 轨道高度分析

(2) 卫星速度分析

卫星速度分析通过分析用户所选卫星在其周期内，卫星速度随时间变化数据。系统对数据进行展示。



图 7-10 卫星速度分析

(3) 星下点分析

星下点分析通过获取当前时间全部卫星星下点坐标，使用热力图展示其分布。

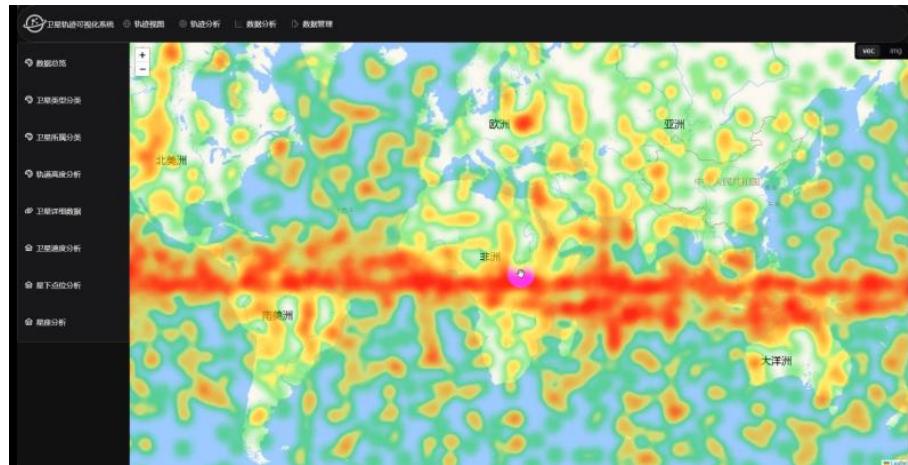


图 7-11 星下点分析

7.1.5 数据统计

(1) 数据总览

数据总览统计和展示本系统中卫星的总体情况，展示卫星总数、TLE 总数、类型总数、类型种类、卫星所属组织总数、具体组织、卫星轨道类型、具体类型、卫星星座总数、卫星星座类型。



图 7-12 数据总览

(2) 卫星类型统计

卫星类型统计用于统计卫星轨道类型、卫星类型并使用饼图展示。



图 7-13 卫星类型统计

(3) 卫星所属统计

卫星所属统计通过统计卫星所属国家（组织或地区）并使用饼图展示。

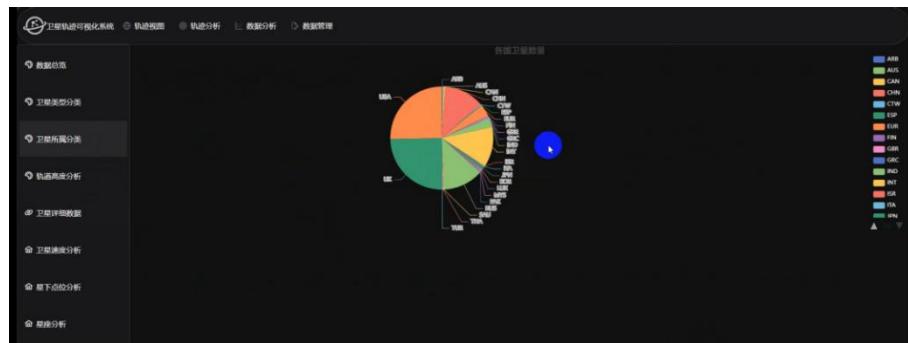


图 7-14 卫星所属分析

(4) 卫星星座统计

卫星星座统计用于统计各个星座卫星数量并使用柱状图展示。



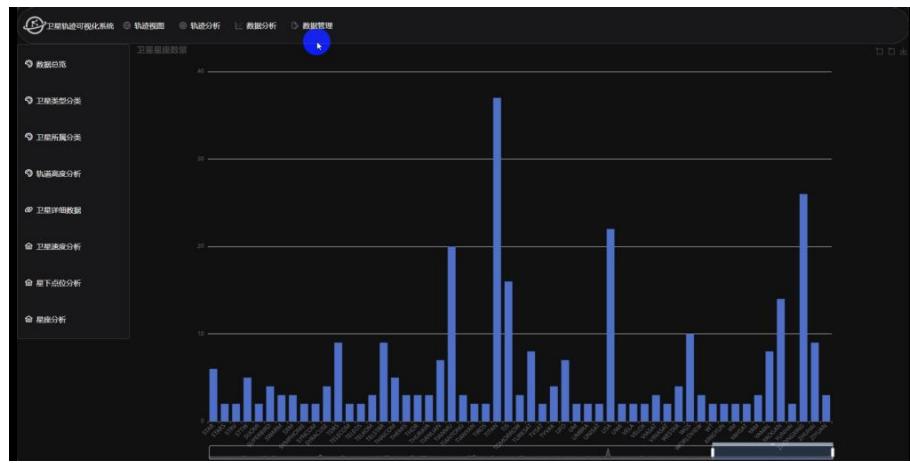


图 7-15 卫星星座分析

7.1.6 数据管理

(1) 删除数据

	id [PK] character varying (50)	name character varying (255)	inid character
1	00001	NewName	NEW123
2	00634	SYNCOM 2 (A 26)	63031A
3	00858	SYNCOM 3	64047A

图 7-16 待删除数据

```
Welcome
$ del --id 00001
{"success": true, "message": "卫星数据删除成功"}
$
```

图 7-17 删除成功

(2) 更新数据

```
Welcome
$ update --id 00001 --year 2020 --tracktype "Lunar Transfer"
{ "error": "卫星不存在: ID=0001" }
$ update --id 00001 --year 2024 --tracktype "Lunar Transfer" --country "测试"
{ "success": true, "data": "更新成功" }
$
```

图 7-18 更新数据

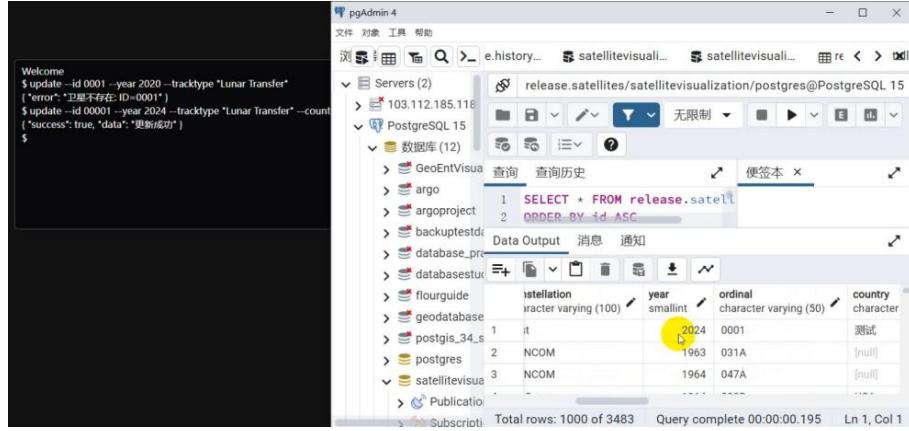


图 7-19 更新成功

(3) 导出数据

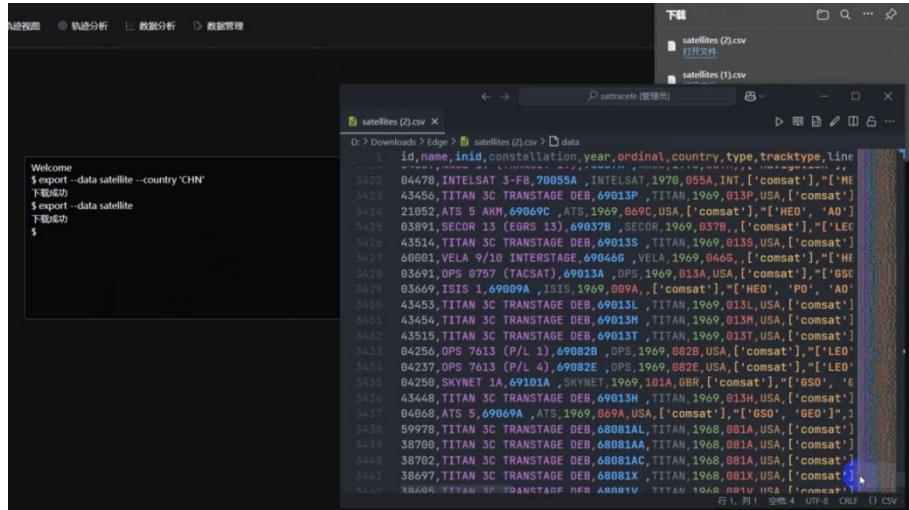


图 7-20 导出数据

7.2 测试结果分析

本系统中所有误差最核心最根本的原因来源于 TLE 数据本身的误差，所有分析精度同样取决于 TLE 数据的精度。这是由 TLE 数据本身特性所决定的，由于目标机动、空间目标碰撞、空间环境改变和错误匹配等事件，TLE 数据的异常值不可避免，TLE 数据本身存在固有误差。学术界相关领域学者已对其进行了充分大量的研究，本节只针对该系统实现所引起的误差、相关结果的精度问题及执行效率进行分析。

7.2.1 技术选型导致的误差分析

SGP4 模型（通过 satellite.js、sgp4.api 实现）未考虑高阶引力场、大气阻力等摄动力，处理不同类型的的空间目标定轨精度各不相同。近地目标定轨精度为百

米量级，SGP4/SDP4 模型预报精度主要与目标的高度有关，对于近地目标，轨道高度越低，预报误差越大；对于高轨目标，轨道高度越高，预报误差越大，而对于椭圆目标，近地点高度越低，预报误差越大。

其坐标系转换算法，ECI 坐标转换 ECEF 坐标依赖格林尼治恒星时（GMST）计算，时间精度不足会引入位置偏差。

7.2.2 轨迹可视化误差分析

Sapcekit.js 设计本意在于进行宇宙尺度的天文可视化，本系统为解决性能与视觉效果问题，引入该库，在展示卫星轨迹时候，因为地球与卫星距离相较于 Spacekit 设计的距离单位（天文单位 AU）太小，导致可视化时视觉效果不佳，为适配 Spacekit，将地球半径和卫星半长轴放大 10^4 倍，导致空间尺度失真（虽保持相对位置，但绝对尺度不真实）。

计算可供 Spacekit.js 使用的轨道六参数时，采用开普勒第三定律（5.3.2 节方法 2）计算半长轴，该方法精度依赖平均运动周期精度，可能影响半长轴结果。

7.2.3 过境分析功能误差和效率分析

过境分析是依赖于时间驱动，时间轴组件当时间戳发生变化时使用 `this.$emit` 方法触发事件，执行相关逻辑。因此其分析精度直接受到事件触发频率（即时间步长）的影响，可能会漏检短时过境事件。

在使用 Turf.js 执行入境判断的代码中，直接影响分析精度的因素在于分析区域的精度、星下点点位精度。Turf.js 中 `turf.booleanPointInPolygon` 内部使用了一种几何算法来判断点是否在多边形内部。虽然 Turf.js 的算法通常是比较可靠的，但在某些极端情况下（例如，点非常接近多边形的边界），可能会出现精度问题。

分析区域多边形的顶点越多，其形状越复杂，计算量也越大。复杂的多边形会导致计算时间增加，通常不会直接影响判断的精度，但是会导致分析逻辑的执行效率下降。

7.2.4 时间离散化导致误差

轨迹采样依赖固定时间步长的相关功能（覆盖分析、卫星速度分析、轨道高度分析等）的精度会直接受到时间步长的影响，从而导致误差，然而时间步大，会导致逻辑执行效率问题。

核心团队

第八章

空天守望者：人造卫星态势感知系统

第八章 核心团队

姓名	面貌	分工模块	主要职责内容
栗铭阳(组长)		轨道计算与可视化	负责轨道数据处理、TLE 解析与轨道推演逻辑实现，利用 Spacekit 进行轨道建模，使用 Cesium 渲染轨迹、星下点与覆盖范围，实现双引擎协同可视化。
李静怡		Web 前端与 UI 交互开发	负责基于 Vue 3 与 PrimeVue 的前端页面搭建，设计并实现用户界面交互逻辑、时间轴控件、图层切换、弹出面板与交互事件绑定，确保页面响应流畅、体验友好。
张梓霖		数据与后端管理	负责后端系统构建，基于 Django 框架设计并实现 RESTful API 接口，完成轨道数据 (TLE) 的存储、版本控制、模拟参数提交与查询逻辑，构建 PostgreSQL 数据库模型。
范思倩		项目集成与测试	负责前后端联调、功能模块集成与部署上线，完成接口测试与可视化同步验证，编写测试用例与部署脚本，撰写项目技术文档并保障各模块协同运行稳定性。

表 1 团队分工表

附件

第九章

空天守望者：人造卫星态势感知系统

第九章 附件

附件一：软件著作

 中国版权保护中心
CHINA COPYRIGHT PROTECTION CENTER

流水号: 2025R1111416646

流水号: 2025R1111416646

软件登记受理通知书

软件名称: 人造卫星轨迹可视化系统软件V1.0

登记类型: 计算机软件著作权登记

申请人: [Redacted]

根据《计算机软件著作权登记办法》第十九条的规定,对申请人提出的上述计算机软件著作权登记申请,中国版权保护中心予以受理。

受理号: 2025R11S1323981

经核实确认中国版权保护中心收到如下申请文件:

打印签字或盖章的登记申请表	4
程序鉴别材料 - 一般交存	60
文档鉴别材料 - 一般交存	70
著作权人(1) - 居民身份证复印件	1
著作权人(2) - 居民身份证复印件	1
合作开发合同或协议	3

中国版权保护中心软件著作权部
2025年06月19日

网址: www.ccopyright.com