密级：　　保密期限：

# 北京郵電大學

# 硕士学位论文


题目：实时与历史结合的多视图
态势呈现系统设计与实现

专　　业：计算机科学与技术

学　　院：网络技术研究院

**2019** 年 **3** 月 **02** 日

## 独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

## 关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

# 实时与历史结合的多视图态势呈现系统设计与实现

## 摘 要

面对如今如此繁多复杂又互有联系的监测手段，物联网系统需要接入多种传感器设备，并有必要综合展示海量的多源异构感知数据。

现有的物联网态势系统可以接入多源异构的传感器设备，但是系统展示的信息缺乏关联度，系统仅提供了多数据源的单一信息，无法展示融合多传感器数据后的综合信息。系统对于目标的认知缺乏深度，仅仅通过单独一两种感知手段管中窥豹，缺少以时空角度出发的目标画像信息展示，难以挖掘监控实时告警信息背后蕴含的更深层次的威胁判据信息，缺乏深入理解各种告警事件、危险行为的本质，导致对区域防护问题的整体认知出现偏差。系统对目标的认知缺乏空间性，无法从空间角度展示目标的区域流动、热点密度等特征。并且现有系统用到的技术仅仅能满足实时监控一种展示模式的需求，这种模式仅仅强调实时展示态势信息，忽视了历史数据的查询、回放，因此研究一种支持实时历史双展示模式的架构是推动态势系统发展的必然要求。

本文设计实现了一个基于时空范式的实时历史双展示模式态势呈现系统。系统接入多源异构感知数据，基于目标信息图谱与多维度数据视图进行时间维度的实时态势监控与历史数据回放，基于 GIS 服务进行空间维度的粗粒度、全局性的目标跟踪判读与目标画像。态势系统仿真实验的结果表明该系统可以有效的展示不同设备数据之间的关联度，以空间角度对目标判读认知，并以实时与历史结合的双展示模式进行态势呈现，且在回放时采用缓冲式批量加载算法实现低延时、高效率的历史回放效果，最终通过功能与系统测试验证本文所设计的系统具有可行性和有效性。

**关键词** 态势显控 物联网 知识图谱 大数据可视化

# DESIGN AND IMPLEMENTATION OF REAL TIME AND HISTORY MULTI-VIEW TREND DISPLAY SYSTEM


## ABSTRACT

The Internet of Things technology has brought a lot of convenience to regional situation monitoring and greatly improved the efficiency of decision-making. Facing complex and interrelated monitoring methods, IoT systems need to access a variety of sensor devices.It is necessary to comprehensively display a large number of multi-source heterogeneous sensing data.

The existing work can access multi-source heterogeneous sensor devices and display the sensing data of different devices on different visual interfaces. However, the information displayed by the system lacks relevance, and it is difficult to clarify the relationship between different device data, and it is impossible to display comprehensive information after merging multi-sensor data. Moreover, there is few work that meet the needs of query and playback of historical data.

The existing IoT situation system can access multi-source and heterogeneous sensor devices and display the perceptual data of different devices on different visual interfaces. However, the information displayed by the system lacks the correlation degree, the data of different devices are completely separated on the visual interface, the system only provides a single information of multiple data sources, it is very difficult to clarify the relationship between the data of different devices. Unable to display integrated information after fusion of multi-sensor data. The system lacks the depth of understanding the target, only through one or two kinds of

perceptual means, only through a single way of perception, the lack of spatio-temporal view of the image of the target information display, unable to explore the surface of the monitoring systemThe deeper information of hidden threat criterion contained in the phenomenon of target threat lacks a deep understanding of the nature of various alarm events and dangerous behavior which leads to the deviation of the overall cognition of the regional protection problem. Because of the lack of spatial characteristics, the system can not display the regional flow and hot spot density of the target from a spatial perspective. And the technology used in the existing system can only meet the demand of real-time monitoring a display mode, which only emphasizes the real-time display of situation information, ignoring the query and playback of historical data. Therefore, it is an inevitable requirement to develop situation system to study an architecture that supports real-time history dual display mode.

This paper designs and implements a real-time historical dual display mode situation system based on space-time paradigm. The system accesses multi-source heterogeneous sensing data, performs real-time situation monitoring and historical data playback based on target information graph and multi-dimensional data view, and performs coarse-grained and global target tracking interpretation and target portrait based on GIS service. . The results of the situational system simulation experiment show that the system can effectively display the correlation between different equipment data, and has a deep presentation of the target threat criterion information. It can also recognize the target from a spatial perspective, and display the situational data in a dual display mode combining real-time and history, and use the buffered batch loading algorithm to achieve low-latency and high-efficiency historical playback during playback. The system designed is feasible and effective.

KEY WORDS situation control internet of things knowledge graph big data visualization

# 目录

# 第一章 绪论

## 1.1 课题背景与意义

当下物联网技术日益发展，海量的多源异构传感器被接入到物联网系统中。以目标为中心的物联网态势系统需要接入多种传感器设备，并有必要综合展示海量的多源异构实时感知数据。此外，海量的目标多维度历史信息被存储并积累，带来了进行历史目标态势展示的需求。目标历史多维信息往往同时具备时间维度与空间维度的属性，这需要物联网目标态势系统同时具备历史数据时间切片快照回放与大规模空间位置渲染的能力。针对物联网目标态势展示的新需求，现有的态势系统暴露出了很多能力盲点与技术缺陷：

(1)现有的物联网目标态势系统展示的信息缺乏关联度。虽然多源异构的设备被接入，但异构设备的数据展示零散化、碎片化，系统仅仅提供了多数据源的单一信息，很难理清不同设备数据之间的关系。同时不同设备的数据之间没有做关联，无法展示融合多传感器数据后的综合信息。

(2)现有的物联网目标态势系统对目标的认知缺乏空间性。系统仅仅是在时间角度对目标进行实时行为特征监控，过分强调目标的时间属性，忽视了目标的历史空间数据中蕴含的信息，无法从空间角度展示目标的区域流动、热点密度等特征。

(3)现有的物联网目标态势系统对目标的认知缺乏历史性。系统仅仅是做单纯的监控，无法将实时监控的结果整合形成目标长期的态势监控信息,并且对于目标长期的态势监控信息缺乏挖掘分析。即使是通过挖掘得到了目标行为特征的少量描述信息，也不能对实时判定目标行为进行数据支持。

(4)现有的物联网目标态势系统展示模式具有局限性。系统用到的技术仅仅能满足实时监控一种展示模式的需求，这种模式仅仅强调实时展示态势信息，忽视了历史数据的查询、回放，现如今极度缺乏一种支持实时历史双展示模式的态势系统架构。

为了解决上述问题，本课题经过调研发现利用实时与历史结合的方法，基于目标信息图谱进行实时态势监控与历史数据回放，基于

GIS(Geographic Information System 地理信息系统)服务进行目标跟踪判读与目标画像。最终形成一个基于时空范式的实时历史双展示模式的目标态势系统。

基于上述背景，本课题定下了如下方案：本课题研究实现一个基于目标信息图谱、实时与历史结合的多视图物联网态势显控系统，系统会将物联网与云 GIS 技术相结合，以目标为核心对多种监测数据进行关联、融合、挖掘分析，从而形成目标信息图谱与多角度数据视图，最终基于时空范式与实时与历史结合的大数据可视化技术实现目标态势可视化。。

该物联网目标态势呈现系统有如下优势：实时监控方面，使用目标信息图谱与多角度视图展示同一监控目标的不同设备数据之间的关系，呈现出将多种手段不同维度的、短期描述信息关联至同一目标下后的目标综合数据，并且展示对目标长期的行为特征进行挖掘分析后得到的告警跃迁过程及威胁判据信息。空间分析方面，系统基于 GIS 服务从空间角度对单个目标进行跟踪判读，展示目标的迁徙、热点、蜂窝等空间历史信息，并结合目标长期运动轨迹与历史行为特征，展示了目标的画像信息。系统架构方面，系统提出一种缓冲式批量数据加载算法，可以从时间角度对历史数据按照全景、区域、事件等模式进行查询、回放，形成一种实时历史双展示模式的系统设计方案，不仅展示了传感层传输至应用层的实时数据，还可以对历史数据按照全景、区域、事件等模式进行查询、回放。

## 1.2 课题研究内容

基于以上背景，本课题以设计与实现实时与历史结合的多视图物联网态势显控系统为目的，以区域防护监控为应用场景，研究使用 GIS 技术、大数据可视化技术对目标进行实时监控、历史回放、跟踪判读、目标画像的技术方案。本课题主要研究内容如下：

（1）系统的整体功能与服务架构设计与构建

系统基于面向多维数据的多级存储系统、物联网接入服务与实时计算与历史挖掘服务，构建出一个实时与历史结合的多视图物联网目标态势展示系统。实时服务与历史服务之间既具备多样化的数据交互又各自有明确的服务边界，实现了一个实时历史服务数据互相生产消

费、内聚化、自治化的面向服务的系统架构。

（2）基于目标信息图谱和多数据视图的目标实时监控与历史回放研究与实现

多设备对区域的监测必然获得同一目标多维度的数据，通过对探测目标的多传感器数据进行数据关联、融合、挖掘、分析后，可以形成存储了情报数据的目标信息图谱及目标感知、行为、特征、威胁数据视图。图形化系统根据融合后的数据与数据分析结果，可提供给使用者多目标的实时感知、行为特征、威胁判据分析数据视图展示，还可以对多目标以全景、区域、事件等方式进行缓冲式批量数据加载的历史回放。如过去一个月某一个特定区域的历史航迹、告警事件、异常记录等。根据目标信息图谱及多数据视图进行详细、必要、友好的实时、历史图形化展示是本课题主要研究内容。

（3）基于 GIS 服务的目标跟踪判读及目标画像研究与实现

同一目标多维度感知、行为、特征数据持久化在用于长期存储的数据仓库中，通过对数据仓库查询可以得到目标在时空维度的历史数据视图，凭借 GIS 技术与大数据可视化技术对目标的时空数据进行跟踪判读，可以提供给使用者关于目标的迁徙态势、热点分布区域、告警蜂窝视图，如目标一个月内的迁徙轨迹、一年内的停留区域热力分布图等。目标画像是将目标行为、特征、事件的差异，将他们区分为不同的类型，从每种类型中抽取典型的特征赋予标识、统计要素等描述形成一个目标原型，即目标信息标签化，并且对这些特征分析统计挖掘潜在价值信息，抽象出一个目标的特征全貌。目标画像是解决监控问题的基础，能够帮助安防监控者找到对的真正需要关注的目标。对目标的跟踪判读及目标画像展示是本课题需要重点研究的内容。

（4）态势呈现系统的实时与历史双展示模式设计与组件化实现

系统使用组件化架构分离实时监控模式与历史回放模式的功能模块与视图表达，并通过组件隔离消费方与服务内部实现细节的绑定，避免了服务对消费方的破坏性修改。通过边界控制实时与历史服务的责任，逐步划分服务上下文，实现实时监控服务与历史回放服务之间的解耦隔离。通过组件信号发布订阅模式暴露实时服务与历史服务的应用程序接口，实现实时监控服务与历史回放服务之间的无缝服务切换、服务交互。通过结合组件化系统与实时监控与历史回放服务，构建除了一个高内聚、低耦合无缝切换的实时历史双展示模式目标态势呈现系统。

## 1.3 论文组织结构

实时与历史结合的多视图态势呈现系统设计与实现共分为六章，章节概要如下：

第一章是绪论，包括了物联网目标态势系统面临的课题背景与课题难点，介绍了本文的研究内容与本系统解决的课题背景提出的现状问题。

第二章是相关技术介绍，主要讲解介绍了本文中使用的基础性技术内容

第三章是态势呈现系统的总体设计，分别设计了整体系统架构，系统功能架构，系统服务架构，系统组件化架构，系统的上下行数据流与系统总体流程。

第四章为态势呈现系统时空范式设计与组件化实现，主要从时间范式和空间范式的角度分别讨论了设计原则，并且讨论了系统的组件化实现。

第五章为系统测试与验证，主要对系统的功能和性能进行测试，从而验证系统具备可行性与有效性。

第六章为总结与展望，总结了本文的工作贡献，并且从群体感知与空间范式的角度对未来的系统发展进化进行了展望。

# 第二章 相关技术介绍

## 2.1 半监督学习与深度置信网络

本系统中采集到的包含标注的目标威胁程度数据数量很少，但是含有大量包含有目标各种行为、事件的未标注数据，故使用半监督学习使用大量未标注数据和少量已标注数据一起来构建更好的分类器的半监督学习方法成为比较好的选择。 经典的半监督学习方法包括自训练、协同训练、三组式训练、再生模型、图算法和直推式支持向量机。

本系统设计了基于主动学习的深度置信网络的半监督学习态势估计模型。深度置信网络是一个包括很多隐藏层的神经网络模型，它层层抽取输入信息的无监督学习和用固定标签微调整个网络的监督学习使得它更加适合半监督学习。图 2-1 给出了 DBN（Deep Brief Network 深度信念网络）的结构样例。这种一层层构建的方法是通过一个叫作限制玻尔兹曼机（Restricted Boltzmann Machines）实现的，它是一个两层循环神经网络，输出层各个结点与输入层的各个结点有无方向的对称性连接。用贪心无监督学习方法逐层训练后，深层架构底层的原始特征被组合成更加紧凑的高层次特征。

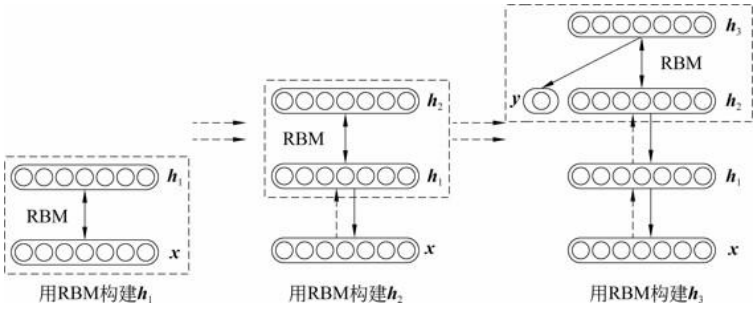


**图 2- 1 深度置信网络结构图**

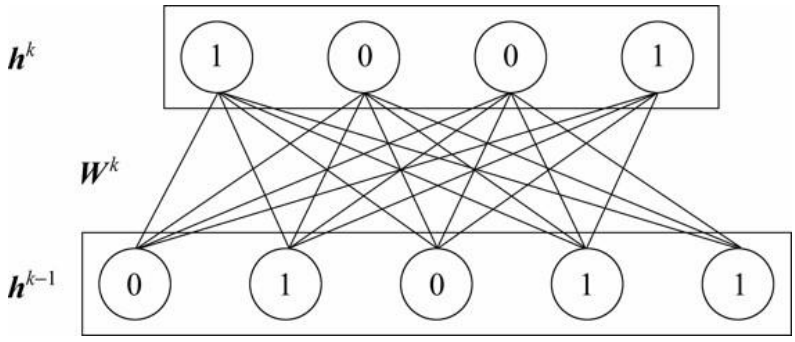


**图 2- 2 限制玻尔兹曼机结构**

## 2.2 目标跟踪微网络模型与卷积层

用于目标跟踪检测识别的卷积神经网络训练的模型为本系统的提供了视频跟踪服务，用于目标跟踪的卷积神经网络中常用的卷积层设计与常用网络如下所示。

### 2.2.1 分组卷积层与深度分离卷积层

分组卷积层的输入按照通道数划分为 g 组，每小组独立分别卷积，结果联结到一起作为输出，使用 Output channel 作为分组卷积层的输入，之后分为三组三乘三的标准卷积层，组内信息通信流畅，不同的分组之间互相独立。

分组卷积层的极端情况成为深度分离卷积层，其分组数量等于输入通道数量，其中每个通道是独立聚集成一部分进行卷积，最终结果聚合各个独立通道间的输出，分组数量等于通道数量，通道之间无信息流动。

### 2.2.2 SqueezeNet 网络与 MobileNet 网络

SqueezeNet 网络模型的特点是微型化，它的设计方法为使用一乘一的标准卷积层代替三乘三的标准卷积层，大量替换复杂卷积层，大大减少参数和计算量。使用 Squeeze 层减少三成三标准卷积层的通道数，进行下采样延迟，保证大激活分辨率。

SqueezeNet block 中第一个一乘一的标准卷积层将输入通道数压缩到 1/8 传输至三乘三的标准卷积层，上下两路标准卷积层扩大四倍后联结起来等于输入通道数量。block 中一乘一的标准卷积层和三乘三的标准卷积层的参数数量之比是

1:3，计算量比是 1:3。

MobileNet 是一个面向移动端的微网络，致力于网络微型化、快速化。它由三乘三深度分离卷积层与一乘一的标准卷积层组成，三乘三深度分离卷积层将三乘三标准卷积层的计算量大幅度减少，一乘一的标准卷积层支持通道间数据交互。结构高效，使用代替标准卷积层的方案牺牲微小性能下降得到数倍效率提升。

网络包含宽度乘子与分辨率乘子参数。宽度乘子参数将所有层的通道数乘一个宽度因数，模型与计算量被压缩为因数的平方倍。分辨率乘子将输入分辨率乘此因数，实现所有层的分辨率变为因数倍，模型大小不变，计算量下降因数平方倍。深度模型要好于宽度模型。

## 2.3 Spring Boot 简介

本系统使用了 Spring Boot 作为服务端，使用轻量级的 Spring 应用初始化与搭建开发过程，不需要复杂的配置模板，使用了大量默认预配置，可以进行快速的服务端迭代开发。

Spring 使用嵌入式的 Tomcat 用约定大于配置的理念进行服务端微框架开发，它使用 starter POMs(Project Objects Models 项目对象模型)提高配置的效率，用一种简洁化的方法搭建项目。服务端不需要 XML 文件配置，而是用 JavaConfig 整合其他开发项目。

## 2.4 MVVM 架构

本系统通过 MVVM(Model-View-ViewModel 模型-视图-视图模型)架构实现前端的视图数据解耦，MVVM 架构是系统组件化的基础。

MVVM 旨在利用 WPF(Windows Presentation Foundation Windows 呈现基础)中的数据绑定函数，通过从视图层中几乎删除所有 GUI(Graphical User Interface 图形化用户界面)代码，更好地促进视图层开发与模式其余部分的分离。不需要用户体验开发人员编写 GUI 代码，他们可以使用框架标记语言，并创建到应用程序开发人员编写和维护的视图模型的数据绑定。角色的分离使得交互设计师可以专注于用户体验需求，而不是对业务逻辑进行编程。这样，应用程序的层次可以在多个工作流中进行开发以提高生产力。即使一个开发人员在整个代码库上工作，视图与模型的适当分离也会更加高效。

MVVM 模式试图获得 MVC(Model View Controller 模型视图控制器)提供的

功能性开发分离的两个优点，同时利用数据绑定的优势和通过绑定数据的框架尽可能接近纯应用程序模型。它使用绑定器、视图模型和任何业务层的数据检查功能来验证传入的数据。结果是模型和框架驱动尽可能多的操作，消除或最小化直接操纵视图的应用程序逻辑（如代码隐藏）。

## 2.5 React 框架与组件化

本系统使用 React 框架借助组件化的思想实现了实时服务与历史服务的隔离，确定了实时服务与历史服务的边界，使他们可以无缝服务切换。

React 构建 UI(User Interface 用户接口)是使用组件化的方式，而不是常见的模板。组件并不是一个新概念，它是某个独立功能或者界面的封装，达到复用或者 UI 和业务松耦合的目的。React 框架里面使用了简化的组件模型，但更彻底地使用了组件化的概念。React 框架通过组件的方式实现视图功能的隔离，实现组件自治、组件隔离、组件解耦。

React 作为一个 UI 框架，不可避免要有界面上元素的交互。为了提高性能，React 在操作页面交互时引入了虚拟 DOM(Document Object Model 文档对象模型)的概念。虚拟 DOM 是在 React 中用 JavaScript 重新实现的一个 DOM 模型，和原生的 DOM 并没有多少关系，只是借鉴了原生 DOM 的一些概念。因为减少了不必要的复杂性，实践校验的结果是虚拟 DOM 的性能比原生 DOM 高很多。基于 React 开发中构建的 DOM 都是通过虚拟 DOM 进行的。在 React 的运行环境中，首先各种不同的视图自适配的渲染了许多数据，当数据状态转移后,DOM 树产生了 DOM DIFF(Difference)变化，最终把 DOM DIFF 变化的部分通过状态转移函数进行更新。React 会在同一个事件循环内合并 DOM 的变化，只是会对比开始和结束的 DOM 变化，忽略中间过程的 DOM 变化。尽管每次数据变化都是重新构建 DOM 树，但虚拟 DOM 的操作性能极高。这样使用 React 时，开发者不在需要关心数据变化时页面上 DOM 元素的更新，而只是关心各个数据状态下页面实际展现的效果。

## 2.6 大数据可视化技术

根据面向的数据类型，可将大数据可视化技术分为两类：基于时间范式的大数据可视化技术，基于空间范式的大数据可视化技术。

### 2.6.1 大数据可视化时间范式

时间属性广泛地出现在各类数据中，例如船舶开始航行时间、微博发布时间等。从可视化技术的视觉角度出发，基于时间范式的大数据可视化方法可被归类为基于位置的时间范式大数据可视化方法和无位置的时间范式大数据可视化方法。

基于位置的时间范式大数据可视化方法使用数据点在时间轴上的绝对位置表示时间信息。时间轴可以是线性的（位置或长度）或是环形的（角度）。在线性时间轴上，每个数据点的绝对位置对应着一个确切的时间。因此，数据点之间的距离（或是轴上一条线段的长度）可以用来表示时间的跨度。类似的，在环形时间轴上，我们可以根据每个数据点所在的弧度确定一个具体的时间，并且用两点到圆心的夹角定义时间片段的长度。如果数据点的位置已被用于表示其他属性，我们还可以选择无位置的可视化方式。例如，当数据点的时间属性呈现离散的特征时，颜色、尺寸、形状及纹理等就会支持对时间属性有效地可视化数据。当数据点的时间属性具有聚集的特征时，可通过区域可视化的方式将时间属性相同的数据点包裹于同一个封闭区域中[1,2,3]。

基于上述基础时间范式大数据可视化方法，可视化研究者提出了一系列针对时间分析的可视化方法。由于时间具有线性和循环的特征，针对时间分析的大数据可视化方法可被大致分为两类 ：线性时间的大数据可视化方法以及周期性时间的大数据可视化方法。

在线性时间的可视化方法中，折线图和堆叠图最为常见。折线图可供用户方便地观察并比较多条时间序列的绝对值，而堆叠图则适用于观察多个随时间变化的序列值及其累加的总和。

周期性时间的可视化常被用于分析具有一定周期性的人类活动数据与城市环境数据。其中，最为常见的一种可视化方法是使用环形时间轴。它将某个时间范围（如 24 小时）按照一定粒度划分，并根据时间属性聚集数据值，最后将这些聚合的数据沿径向布局。比如，径向上每一个圆环颜色的深浅表示船舶在一天不同时间内的活动热度。周期性时间的大数据可视化在大数据可视分析中已有广泛应用。

### 2.6.2 大数据可视化空间范式

空间属性是城市数据的重要组成部分。针对空间属性的可视化技术分为三种 :基于点的可视化技术、基于区域的可视化技术以及基于线的可视化技术。

基于点的可视化技术能让用户直观地看到数据所蕴含的位置信息。基于线的

可视化方法通常用于分析基于交通网络的城市大数据，例如交通轨迹数据。一种比较常用的轨迹可视化方法是按时间顺序顺次地连接轨迹记录中的采样点，并在连接数据点的线段上借助线段的颜色或粗细等视觉方法对额外的信息进行展示。

基于区域的可视化技术能够在一定程度上解决上述视觉混淆问题。在基于区域的可视化中，我们将空间按照显式规则进行划分（如行政区域、城市功能区域），并且将数据点的非位置属性通过位置属性使其聚合。同时，我们也可以应用许多可视化方法对区域之间的流动进行展示，这将有助于观察区域内部和区域间的数据特征。然而，当区域分布密集且区域之间流动关系复杂时，使用线段对区域间流动进行编码的方法会因为线段互相遮挡而导致严重的视觉混淆。我们可以采取边捆绑、基于矩阵的可视化方法以及基于图符的可视化等技术，在不影响位置信息表达的基础上对区域间的流动数据进行清晰的展示[4,5,6]。

## 2.7 NoSqL 数据库

本系统中面向多维数据的多级存储系统运用了 NoSQL(Not Only SQL)数据库。

### 2.7.1 列式数据库

列族存储是根据 Google 的 Bigtable 建模的，该数据模型是基于稀疏表，它的行可以有任意多列，并且包含提供自然索引的键。列族常用的构件有四种，最简单的存储单元就是列本身，包括一个名称值对。任意数量的列可以组成一个超级列，它给一组排好序的列一个名称。列存储在行里，当行只有列的时候，它被称为列族，当行包括了超级列的时候会，它被称为超级列族。

列族数据库是聚合存储，它不适合进行大规模相关关联数据的查询。它的每一行都代表了一个特定的而不可或缺的实体。每一行提供嵌套的哈希结构。

### 2.7.2 图数据库

一些图数据库使用的是原生图存储，专门为存储和管理图设计优化过的。但是并不是所有的图数据库都使用原生图存储，其中有一些是把图数据序列化之后保存到关系数据库中去。原生的图数据库并不是很依赖与索引，因为图本身提供了一个天然的邻接索引，在原生图数据库中，附在节点的联系自然提供了直接的关联到其他我们有兴趣的节点上。图查询利用这个特性去遍历图，使得操作的执

行效率及其高，每秒钟可以遍历百万级别的节点。

## 2.8 本章小结

本章介绍了系统中设计实现涉及到的相应技术的基本知识，对于其基本的内容做了回顾，为之后介绍总体设计与实现做了理论上的准备。NoSQL 数据库即为面向多维数据的多级存储系统的准备，React 框架、组件化、MVVM 架构即为态势呈现系统设计的基础准备。目标跟踪网络模型与半监督学习，为之后的视频目标跟踪与态势估计模型的训练打下了理论基础。大数据可视化为本文的总体时空范式设计模式提供了理论基础。

# 第三章 态势呈现系统的总体设计

本章将从整体的角度对态势系统的架构、服务、功能、组件、数据流、总体流程做出设计。

## 3.1 系统架构设计

### 3.1.1 系统架构概述

系统整体架构包括物联网感知设备、数据服务平台、面向多维数据的多级存储系统、态势呈现系统。系统架构如图 3-1 所示。

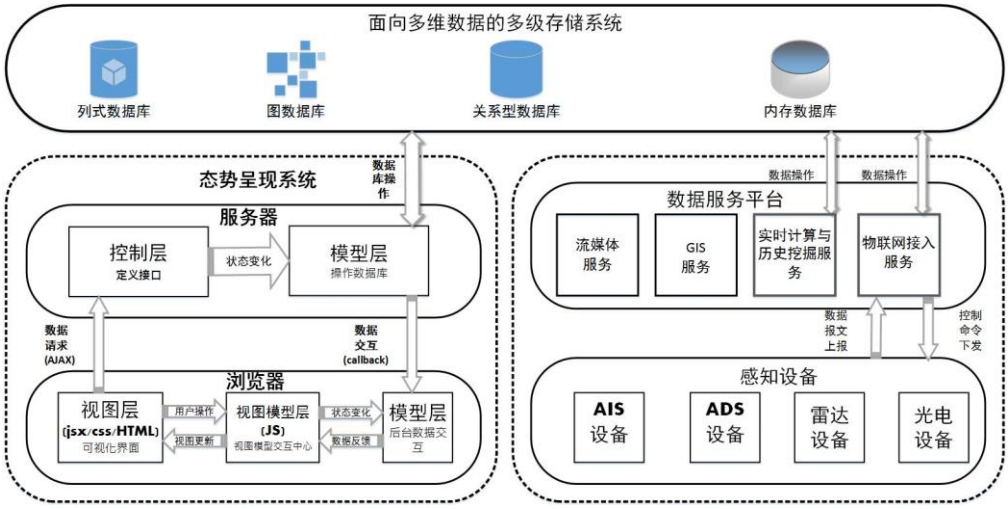


**图 3-1 实时与历史结合的多视图态势呈现系统架构图**

本课题重点研究实时与历史结合的多视图态势呈现系统基于全天候的安防目标实时监控、目标识别跟踪、特征行为提取、行为威胁分析的应用场景，实现了基于目标信息图谱的实时态势监控与历史数据回放,并且基于 GIS 服务对目标跟踪判读及目标画像。

该方法涉及到多层级的计算、存储、可视化技术：在目标设备监控与感知层级，采用范围大、精度低、功耗小的传感设备与回传快速、功耗略大的光电跟踪设备对监测范围内所有目标进行感知，并通过物联网接入服务将协议解析后的感知数据净荷上报至存储系统。

在目标特征提取与威胁分析层级，采用一种离线与实时结合的目标累积识别框架挖掘目标的深层次增值信息与富语义情报信息，通过一系列特征提取与行为分析算法，形成目标信息图谱，并且通过显式规则判断或深度学习模型训练得到目标威胁分析模型，最终按照数据处理程度由低到高形成目标感知、特征、行为、威胁多角度数据视图；

在目标态势可视化层级，采用组件化与大数据可视化技术，以实时与历史结合的方法展示对于多传感器的信息进行关联、融合、挖掘、分析的结果。对多传感器进行数据挖掘的结果包括了目标信息图谱及多角度数据视图。并且展示出对目标跟踪判读及目标画像的具备潜在价值的富语义内容[7,8,9]。

我们通过物联网数据接入服务完成数据净荷的接入，通过数据服务平台完成目标特征提取及行为分析，通过多级存储系统存储目标信息图谱及多角度数据视图。最终通过态势呈现系统将上述各层级的任务结果统合整理在一起，以实时与历史结合的方法，完成目标实时监控、识别跟踪、特征行为提取、行为威胁分析和态势可视化。系统结构如下：

1）物联网感知设备

物联网技术给区域态势监控带来了很多繁多复杂又互有联系的监测手段，系统需要接入多种传感器设备形成海量的多源异构感知数据，这些数据是构成物联网系统的基础，包括了目标的方位、经纬度、径向速度、切向速度、距离等原始感知信息，传感器设备是信息的生产者，产生的信息是数据分析、计算、挖掘的原材料。设备生产出原始感知信息后，通过物联网接入服务进行协议管理、数据净荷提取、报文分发等过程上报至多级存储系统，并且用户可以根据实时上报数据情况通过物联网接入服务对设备进行控制命令下发，执行具体的区域防护任务。

2）数据服务平台

数据服务平台是一个分布式综合协同计算系统，包括了物联网接入服务、实时计算与历史挖掘服务、GIS 服务、流媒体服务。

物联网接入服务实际上是一个分布式的传感器接入及资源管理中间件，感知设备接入后，首先配置基本参数，然后完成协议封装、协议解析、协议适配，接入服务会暂时缓存数据，并使用基于优先级的任务调度方法对缓存感知数据进行持久化，最终打通传感层与应用层，实现感知资源接入，获得多源异构的原始感知数据。

实时计算与历史挖掘服务是一个基于实时计算与离线挖掘的目标累积识别框架，实时计算部分对实时的原始感知数据进行时空配准、轨迹关联、数据融合，

首先发现多种不同的传感手段探测的到的同一个目标的多源异构数据的相关关联关系，并将同一目标的多种手段不同维度的感知数据关联至同一个目标下。然后对整合后实时计算结果形成的目标长期运动轨迹、历史碎片化数据进行并行化的离线挖掘分析，通过特征提取及行为分析算法得到目标本身的属性特征信息与目标行为信息（经过区域、篡改伪装次数、告警行为等），对于提取的目标行为特征信息通过一系列人可理解的显式规则判断与机器通过深度学习方法训练的隐式威胁模型得到目标的威胁分析数据视图，上述计算过程经过数据累积与反复迭代过程最终形成含有目标感知、特征、行为、威胁数据视图的目标信息图谱。实时的信息作用于历史数据挖掘，历史挖掘得到的目标信息图谱又能反过来对目标实时监控、计算、分析、可视化进行服务，该框架是一个用实时与历史结合的思想构建的双计算模式迭代式目标累积识别系统。

流媒体服务是一个接入了光电视频设备并具备对目标的视频监控、实时跟踪、目标识别检测的视频服务系统。它可以实时接入多种光电设备并存储监控视频文件，采用反向代理的方式提供实时视频流，还可以对视场内框选出的具体目标进行目标检测跟踪。

GIS 服务是一个整合了离线地图发布、地理空间数据渲染、点线面图层数据可视化的本地化服务工具。它可以使用 WMTS（Web Map Tile Service 瓦片地图服务）的方式进行瓦片地图服务发布，在浏览器界面呈现不同坐标系标准下的 GIS 地图，并且可以将目标的感知视图中蕴含的位置、方向、速度等信息渲染在地图上，并且能结合大数据可视化的技术呈现迁徙、热点、蜂窝视图，辅助于整体呈现目标区域综合态势。

3）面向多维数据的多级存储系统

本系统采用一套物联网数据多维数据多级存储方案，能够存储设备感知数据从生产到消费过程中衍生出的一系列感知、特征、行为、威胁数据视图，还能使用原生图存储的方法存储目标信息图谱，不仅仅存储了具体的数据元素，还存储了元素之间的关联，使用免索引邻接的方法确保遍历关联数据的效率。

在本存储系统方案中，第一级存储使用关系型数据库存储物联网接入服务的感知数据解析结果。当接入服务中的传感适配器将数据净荷发布至消息中间件后，由于接入服务的持久化模块订阅了所有传感器解析结果的主题，持久化模块会请求加入关系型数据连接池，最终完成原始感知数据的持久化。也为实时数据监控中的设备显控、原始感知数据视图提供底层数据支持，更为实时计算与历史挖掘服务提供了消费数据原材料，以便于特征提取与威胁行为分析。

第二级存储使用支持大规模并行化数据处理的列式数据仓库，实现分布式

存储，并且具备了高可用性、可伸缩性、强扩展性。由于列式数据库具备更好的数据增添特性，可以更加灵活的存储以目标为中心的数据，按照每个融合归一化目标存储为一张表的方式，经过数据关联、挖掘得到的新目标可以灵活的存储在列式数据库中作为多维度数据的存储数据库，对列数据没有严格的限制，即可以不断扩展行的内容，亦可以随时增添新的列，实现横向纵向的数据表扩展，单个多维度目标融合信息表的内容可以达到数十几亿的行的规模，符合态势呈现系统对于实时与历史结合的双展示模式的需求，更有利于存储实时计算与历史挖掘服务得到的目标特征、行为、威胁数据视图的需要。

第三级存储使用原生图数据库免索引邻接的方法存储目标信息图谱，并构建目标特征、行为、威胁数据视图。原生图数据库中的免索引邻接意味着关联节点在数据库里是物理意义上的指向彼此，会在遍历查询大规模关联数据时带来巨大大的性能优势，采用图的方案，性能可以提升一个甚至几个数量级，比起聚合批处理，它的延迟也小很多，因为图数据库的查询执行时间只和满足查询条件的那部分遍历图的规模大小而不是整个图的规模成正比，然而传统的关系型数据库随着数据集的不断增大，处理 join（join-intensive）查询的性能也随之变差。图数据库可以很好的存储特征提取、行为分析算法的结果，并根据威胁分析模型预测的出的威胁分析结果及判据迭代式的构建目标信息图谱，更加灵活的满足目标累积识别和历史数据梳理机制的需要。

4）态势呈现系统

实时与历史结合的态势呈现系统基于 B/S（浏览器/服务器）模式实现。

本系统的服务端指运行在轻量级 web 容器中的后台服务。它使用 Spring Boot 框架进行部署，并且使用 Spring Framework 框架体系中的 Spring MVC 作为核心运行模块。后端使用 ORM/OGM 映射的方式，将目标信息图谱与多角度数据视图中的图数据与批量视图数据映射至后端模型层，并且通过后端控制层定义的接口，将批量数据加载至浏览器端。

系统的浏览器端采用组件化的架构，基于 React 框架并使用 Ant-design 作为 UI 设计语言构建出一个可无缝切换的实时历史双展示模式的可视化界面。浏览器端采用 MVVM 模式分离视图与模型，视图与模型之间具备了明确的边界从而实现视图自适配批量数据的特性。MVVM 使用绑定器、视图模型和任何业务层的数据检查功能来验证传入的数据。结果是模型和框架驱动尽可能多的操作，从而消除或最小化了直接操纵视图的应用程序逻辑。

### 3.1.2 面向多维数据的多级存储系统

本文设计的存储系统实际应用于监控区域周边的陆地、海域状况，多源异构的传感器设备,例如温湿度传感器、AIS 船舶自动识别系统（Automatic Identification System）,生产出原始感知数据后,将原始数据报文通过 TCP、HTTP、UDP 等不同的传输方式将数据接入至系统内，并通过物联网接入服务经过协议管理、协议热插拔、协议解析等技术手段提取数据净荷，然后存储解析后的数据至关系型数据库 MySQL 中。

原始感知数据经过实时计算与历史数据挖掘后,将每个目标的不同维度的数据整合关联到了一起，并且形成目标长期运动轨迹、目标本身的属性特征信息与目标行为信息（经过区域、篡改伪装次数、告警行为等），将目标的轨迹信息、特征行为数据视图等数据关联整合后的信息存储在非关系型列式数据库中。

对于提取的目标行为特征信息通过一系列显式规则判断与深度学习方法训练的模型进行威胁分析，随着各类感知数据的不断累积，利用上述数据关联算法增量式的更新,上述计算过程经过增量式的数据叠加和历史数据梳理机制最终整合为含有目标感知、特征、行为、威胁数据视图的目标信息图谱，将最终生成的目标信息图谱存储至图数据库中，对于新关联上的感知数据索引，需要在图数据库中进行添加，而对于前期由于信息不完全导致的错误关联，则需要在图数据库中进行删除，当各类感知数据累积到一定阈值后，系统将不断的更新图数据库中的多角度数据视图[10,11,12,13]。

本文采用模块化的设计思想进行存储系统架构设计,每个模块具备问题针对性,注重模块之间的关联性、统一性,本文设计的多层次存储系统包含一下模块：

（1） 原始感知数据存储模块

原始感知存储模块主要为设备显控与联动提供支撑性数据，并且是提取目标特征行为，进行威胁分析的原材料，是整个多级存储系统的起点。模块存储了多源异构传感设备上报的数据净荷，并支持快速数据事务处理，为后续模块提供原始感知数据材料。

（2） 数据同步整合模块

数据同步整合模块是原始感知数据存储模块与多维度数据存储模块的纽带,,它可以将原始感知数据模块存储的数据同步至分布式存储集群,在不给原始感知存储事务造成巨大压力的前提下保证较高的数据传输速度。并且完成对原始数据的清洗，对于无效数据、垃圾数据采用不同步，不整合，节省资源的原则。该模块还将各类零散化的原始感知数据抽取整合成为目标维度的完整融合数据,并在进行区域维度的划分，具备整合性、针对性、高效性。

（3） 多维度数据存储模块

多维度数据存储模块存储了经过数据清洗、整合、挖掘后的目标多维度特征、行为、威胁数据视图，具备可伸缩、灵活的存储模式，适应实时监控、历史回放、跟踪判读对于大规模多维数据查询的要求。采用列式数据库存储经过对原始感知数据关联、整合后的以目标为中心的多维度数据视图，按照每个融合归一化目标存储为一张表的方式，可以更加灵活的存储以目标为中心的数据，并可以在数据表内进行横向与纵向的扩展，单个数据表的规模达到十几亿行，几百万列，符合态势呈现系统对于实时与历史结合的双展示模式的需求，更有利于存储实时计算与历史挖掘服务得到的目标特征、行为、威胁数据视图的需要。

（4） 目标信息图谱存储模块

目标信息图谱存储模块使用原生图数据库免索引邻接的方法存储目标信息图谱，并构建目标特征、行为、威胁数据视图。图数据库拥抱联系，原生图数据库中的免索引邻接意味着关联节点在数据库里是物理意义上的指向彼此，其中的细粒度联系与通用联系更适合迭代图计算与增量开发，更加适合图数据模型的平稳演化，不断的发现新的目标与新的维度特征行为视图，形成新节点与新联系图。图数据库可以很好的存储特征提取、行为分析算法的结果，并根据威胁分析模型预测的出的威胁分析结果及判据迭代式的构建目标信息图谱，更加灵活的满足目标累积识别和历史数据梳理机制的需要。

图 3-2 详细展示出存储系统功能模块划分。

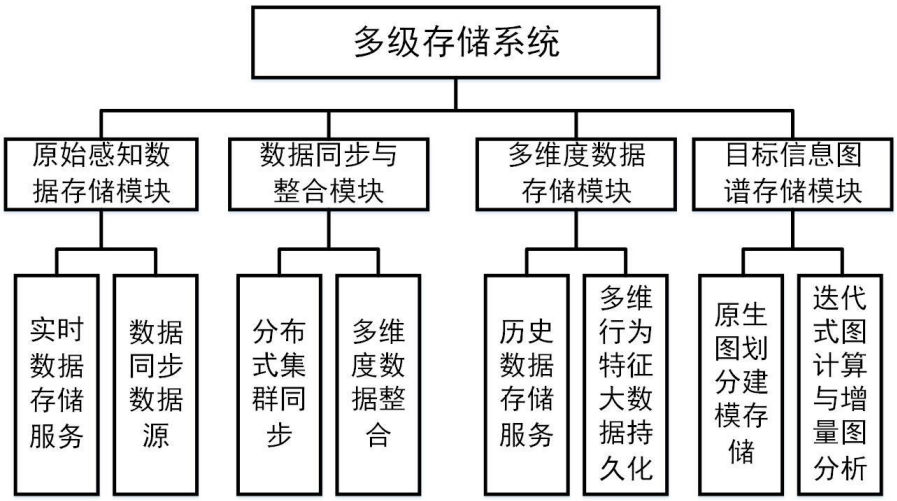


**图 3-2 多级存储系统功能模块**

根据存储系统的功能模块划分设计的多级存储的整体技术架构如图 3-3 所示。

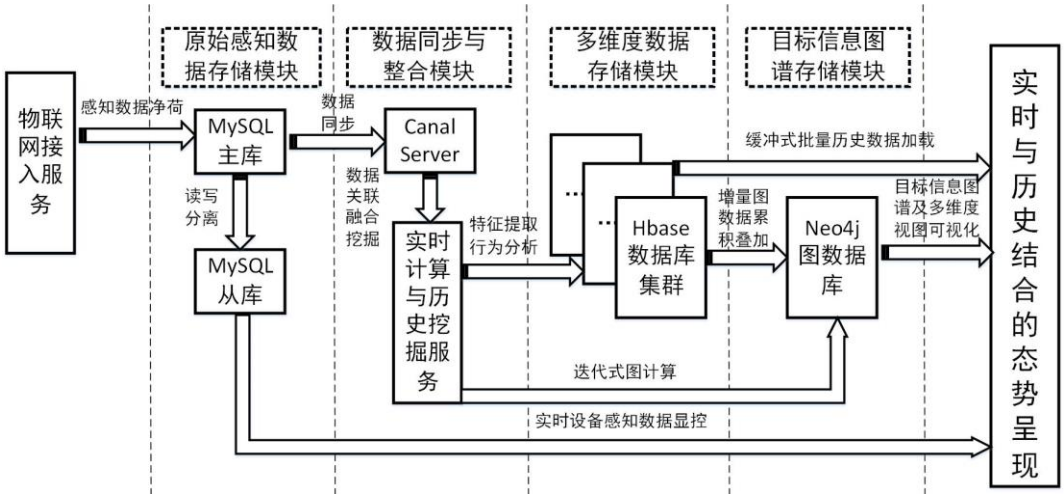


**图 3-3 多级存储系统整体技术架构**

在原始感知数据存储模块使用 MySQL 存储数据，并且采用主从架构实现实现读写分离，使用 Canal 实现数据同步，利用实时计算与历史挖掘服务实现目标维度与区域维度的数据关联、融合，并进行特征行为分析与威胁分析模型训练，将数据关联、融合、分析、挖掘的结果存储到 HBase 数据库集群中，用来为历史数据回放及跟踪判读提供缓冲式的批量时空历史数据加载服务。当各类设备的积累的感知数据索引累积到一定数值后会触发目标特征发现与行为分析算法，从而

构建出了原始目标信息图谱，目标信息图谱存储在图数据库 Neo4j 中，其构建过程需要多次的迭代，是经过历史感知数据增量式的叠加与历史数据梳理机制迭代形成的，不断的将图谱片段整合、完善，删除前期由于信息不对称导致的错误关联，从而降低威胁行为分析中的非对称风险。

为了灵活的存储多维度多区域的目标融合信息，本文采用了 HBase 非关系型数据库。HBase 是一个面向列的分布式存储系统，具备实时读写性，对于超大规模数据集随机访问具有良好的支持。HBase 底层采用键值对的方式进行存储且并不严格控制数据的列信息，可以针对目标的多维度信息进行行与列的扩张，单表可以扩展至数十亿行数百万列的规模。

HBase 最为存储介质主要分为两部分任务，第一部分是针对以目标为中心的存储需求，存储所有目标出现的地理信息及特征行为属性信息，如目标长度、宽度、经纬度、航向、设备警报信息等。数据库以时间为基本单位划分目标的运动状态，以目标唯一特征标识为行键值对，以时间列区分不同时段的状态，时间列由目标运动轨迹的起始时间戳确定。第二部分是存储区域维度的信息，数据库预先针对监控区域做出多关键区域划分，针对每个关键区域设计一个存储表，存

储区域内出现过的所有目标的时空感知信息。

Neo4j 是一个原生图数据库，原生图数据存储结构天生就是可扩展的，使用 Neo4j 建模目标信息图谱后还可以对已经存在的图谱结构添加不同种类的新联系、新节点、新标签、新子图，而不用担心破坏已有的查询与图谱功能，通过感知数据增量式的叠加与迭代式的图计算，一个目标的特征、行为、威胁分析视图数据会越来越丰富，不断完善目标信息图谱中的细粒度联系与通用联系，图规模越来越大。

Neo4j 使用免索引邻接引擎，具备原生处理能力，其中每个节点都会维护其对相邻节点的引用。因此每个节点都表现为其附近节点的微索引，这比使用全局索引的代价小很多，意味着查询时间与图的整体规模无关，它仅仅和所搜索的满足查询条件的部分遍历图的规模成正比，这表明图数据库提供了应对大规模相关关联数据查询的高效解决方法。

### 3.1.3 接口设计

本节将论述态势呈现系统服务端接口的设计方案。后端接口是连接浏览器端与多级数据存储系统的桥梁，通过后端接口将实时的目标多维度数据视图与目标信息图谱传输到前端，为实时态势监控模式提供了数据支持，并且在历史数据回放模式中，通过接口控制大批量数据预加载，并实现缓冲式的批量数据加载、渲染、回放，在跟踪判读及其目标画像任务中，后端数据批量传输的目标大批量空间感知信息为大数据可视化提供了基础数据集。

本系统的接口采用 Spring Boot 框架集成 Spring Data Neo4j 与 Spring Data Hadoop 的方案。实现使用 POJO（Plain Ordinary Java Object）建模大规模图数据、列族数据映射与嵌入式驱动，并且通过 SDN/SDH 提供的存储库接口调用标准 CURD(Create Update Retrieve Delete)方法灵活的操作 HBase 与 Neo4j。

服务端操作图数据的接口使用 Spring Data Neo4j 实现，它使用 OGM（Object Graph Mapping 对象-图映射）将域对象与图数据进行相互转换。使用这种转换机制，我们在对对象进行建模时，只要使用一些简单的注解，就可以让对象与图数据建立起映射关系。SDN 提供了对 OGM 映射的智能管理机制，增强了访问数据库的性能。这主要表现在两个方面：一方面，当一个应用在使用对象并且进行修改的过程中，并不需要直接对数据库进行操作，只有当保存对象时才连接数据库，另一方面，在保存一个对象时，其他与这一对象相关联的对象也能相应地得到保存。

服务端使用 SDN 的存储库接口实现持久化和进行数据接口访问设计。通过简单继承 SDN 的存储库接口，就可以执行标准的 CURD 操作，同时还可以按接口的规范标准自定义方法，实现像使用查询语言一样的查询设计，所有的方法将由 SDN 智能实现。在接口中可以通过注解使用自定义的 Cypher 查询语句。这也是由 OGM 的映射机制实现的，并且通过 OGM 优化处理使查询具有很高的效率，即提供了很好的性能表现。通过 Cypher 查询语言，可以设计出复杂的查询接口，以支持各种各样的业务需求。SDN 还提供了隐式事务管理机制，即对数据库的每一项操作，包括查找数据、保存数据等，都可以使用隐式事务管理。通过隐式事务管理，每一个事务都是自动提交。

服务端操作列族数据库的接口使用 Spring Data Hadoop 实现。接口主要由模板类和 DAO（数据访问对象）类组成。接口对数据库表进行持久化层建模，使用模板类完成 HBase 的配置及初始化工作，并且可进行列簇信息查询及回调信息显示。使用数据访问对象类进行增、删、改、查等数据库操作，可以插入表中指定行，指定列簇，指定列中数据，服务封装后的数据访问对象接口可以处理具体业务逻辑，最终实现批量请求缓冲数据、空间轨迹位置数据、多角度视图的服务需求。

### 3.1.4 组件化通信机制及状态转移

本系统使用组件化思想实现实时历史结合的双展示模式态势呈现，组件以相互之间的关系进行分类主要包括独立组件与继承组件，继承组件是指组件内部的基本元素是从其他外部组件继承得来的，继承组件与外部其他组件之间具备继承关系。独立组件内部的元素相对于其他外部组件是独立互斥的，无其他任何组件相关关联关系。

继承组件之前具备一种继承化的通信机制，可以通过这种组件间的继承关系进行组件状态、属性、数据、函数通信，具备继承关系的组件内部数据流存在两种基本模式：内外层组件数据流模式与平行嵌套组件数据流模式。

继承组件通信机制的第一种模式是内外层组件数据流模式，内部组件通过继承关系将外层组件的属性、原型、状态、函数、状态统一继承传递至组件内部，实现由组件外部向组件内部的数据流动，外部组件通过内部组件运用继承关系对于外部组件状态转移函数的调用，实现由组件内部到组件外部的数据流动。

继承组件通信机制的第二种模式是平行嵌套组件数据流模式，当两个组件的同时成为同一个外部组件的内部组件时，这两个组件互相成为平行嵌套组件，由

于两个平行嵌套组件有一个公共外部组件，平行嵌套组件可以通过调用外部组件的状态转移函数修改外部组件的状态，外部组件经过状态转移后运用继承关系最终将数据、属性、状态、函数等参量最终传输至另一个平行嵌套组件中，实现平行嵌套组件之间的数据流动。

组件化继承通信机制如图 3-4 所示。

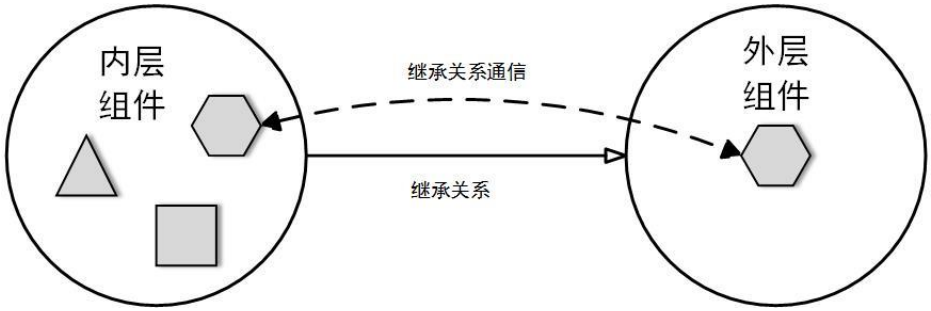


**图 3-4 组件化继承组件通信机制**

独立组件具备高内聚、低耦合的性质，组件之间不存在继承关系，组件与组件之间通信需要借助信号广播器等第三方通信组件，首先发布消息的组件通过组件内部的状态转移函数、触发器发布含有一定主题的信息数据报，信号广播器接收到此消息后向所有连接至此信号广播器的独立组件进行广播通信，信号靶向独立组件通过订阅该主题的信息提取相应主题的数据报净荷，实现独立组件之间的数据流动。独立组件间的通信机制如图 3-5 所示。

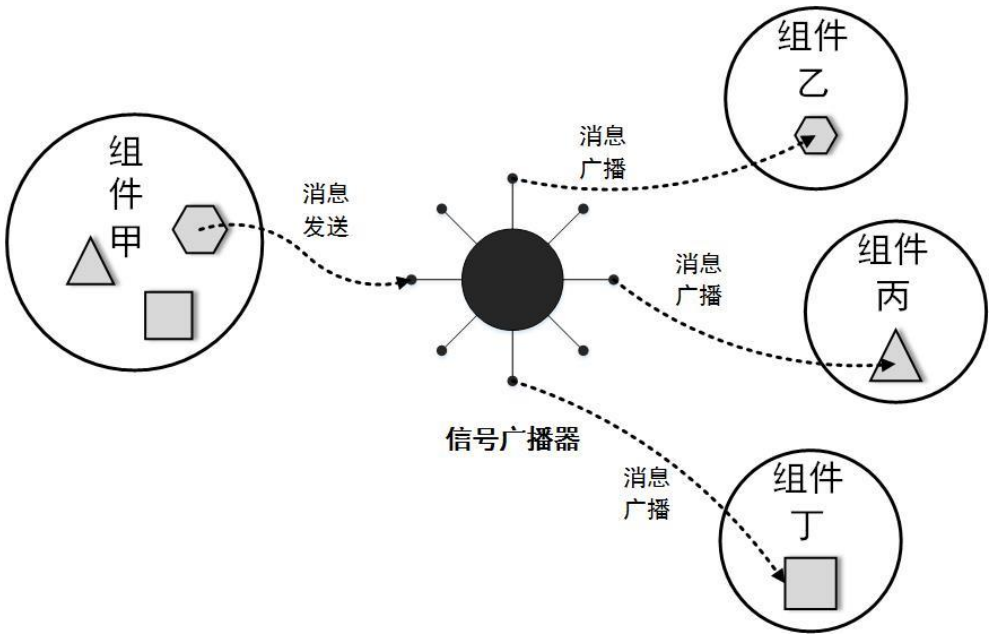


**图 3-5 组件化独立组件通信机制**

组件化可以实现低耦合、高可用的原因是组件之间是功能独立的逻辑单元，每一个组件内部都包含一个微型状态机，组件状态机中的状态转移机制将组件视图、组件数据与组件逻辑联系起来，最终完成组件的核心功能。组件化的状态转移机制如图 3-6 所示[14,15,16,17]。

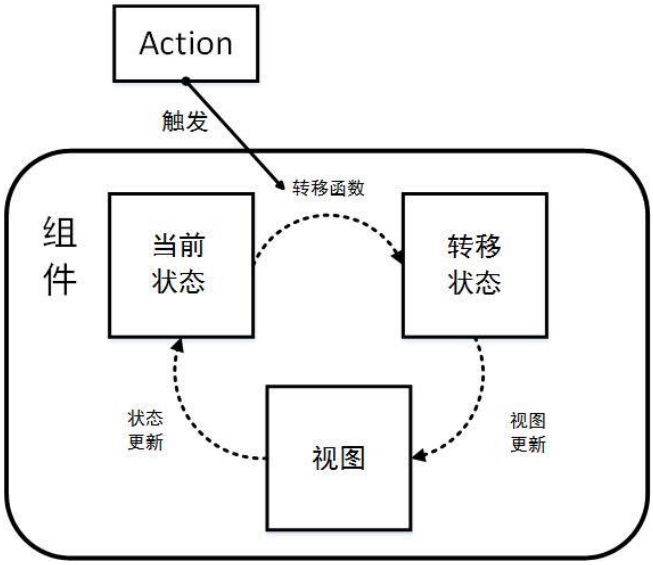


**图 3-6 组件状态转移机制**

用户对组件产生的操作通过触发器触发了转移函数，转移函数根据当前组件的状态计算产生得到组件的转移状态，并且产生转移状态数据，数据经过逻辑处理后引发视图更新，视图更新后当前组件状态产生改变。组件中所有数据的生产起点都是逻辑处理，组件内部一个视图需要一个逻辑处理区实现交互操作，因此逻辑处理区与视图是一对一的关系，一个视图更新、交互也只处理一部分数据，因此视图、逻辑、数据在状态机的转移机制下是一对一的关系，这种组件内部一一对应关系大大降低了耦合性、提高了内聚性。

组件化通信机制具备单向数据流的特性，数据流向清晰可控，由状态转移机制控制，故组件化是一种易抽象、可配置、状态可控、高复用的架构。

## 3.2 系统功能结构设计

实时与历史结合的多视图态势呈现系统基于全天候的安防目标实时监控、目标识别跟踪、特征行为提取、行为威胁分析的实际需求，实现了基于目标信息图谱的实时态势监控与历史数据回放,并且基于 GIS 服务对目标跟踪判读及目标画像。

系统的实现了面向目标的安全态势管控，用实时与历史结合的方法实现多

层次的目标计算、存储、可视化。系统的功能结构以纵向区分为由低到高三个层次：目标设备监控与感知层、目标特征提取与威胁分析层、目标态势可视化层。

在目标设备监控与感知层级，系统利用物联网接入服务将协议解析后的感知数据净荷上报至存储系统，并且使用光电跟踪设备对监测范围内所有目标进行感知；在目标特征提取与威胁分析层级，系统挖掘目标的深层次增值信息与富语义情报信息，通过特征提取与行为分析算法，形成目标多角度数据视图；并且通过显式规则判断或深度学习模型训练得到目标威胁分析模型，最终采用一种离线与实时结合的目标累积识别框架形成目标信息图谱，并增量式的更新目标感知、特征、行为、威胁；在目标态势可视化层级，通过大数据可视化技术实现对目标的跟踪判读与目标画像，以实时与历史结合的方法展示对于多传感器的信息进行关联、融合、挖掘、分析的结果。

态势呈现系统功能结构设计图如图 3-7 所示。

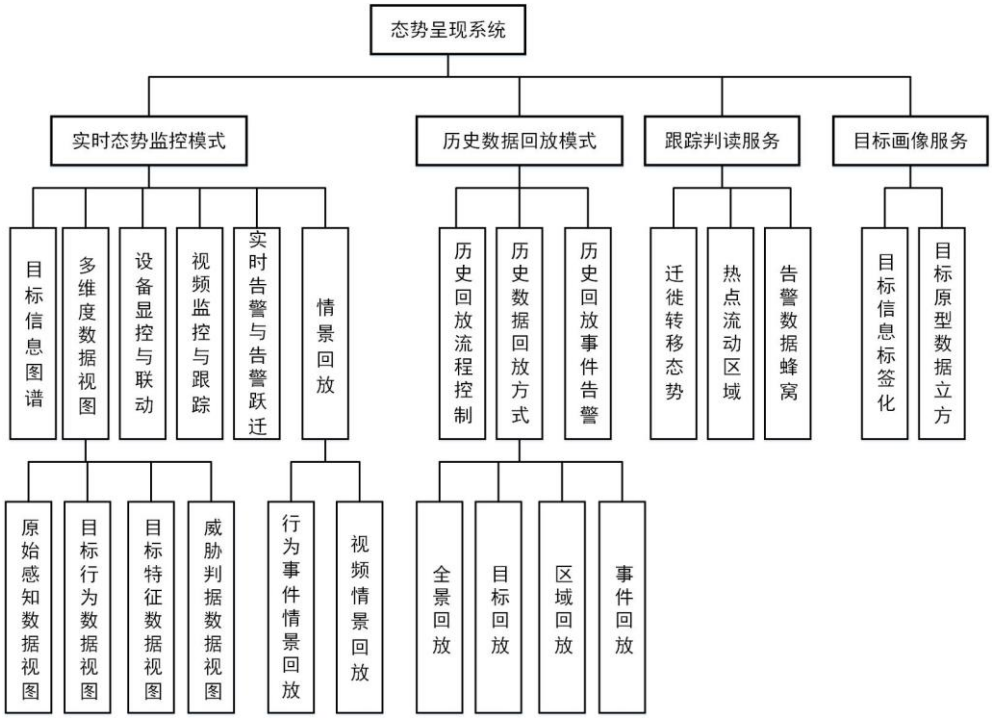


**图 3-7 态势呈现系统功能结构图**

系统从横向的功能上划分，主要分为四个部分：实时态势监控模式、历史数据回放模式、跟踪判读、目标画像。实时监控与历史回放模式是用可无缝切换的实时历史双展示模式呈现目标多维度数据视图与目标信息图谱，遵循时间维度范式。跟踪判读与目标画像服务是将目标的历史空间位置数据使用 GIS 服务以网

格聚合、迁徙、热力等可视化方法，综合呈现目标数据立方，将目标画像化，遵循空间维度范式。

实时态势监控模式展示的内容主要分为两类，第一类为设备实时显控、联动、告警、跟踪、识别的信息。针对设备动态、静态数据进行实时同步刷新的动态探测信息展示与实时航迹绘制、探测范围渲染。第二类为实时的展示历史数据分析的结果，目标实时监控设备形成的目标长期运动轨迹经过挖掘分析、特征提取、行为模式分析后，得到目标特征、行为、威胁的描述信息与知识图谱，为实时目标的检测、识别与总体威胁态势的呈现进行服务。该模式体现了实时与历史结合的核心设计思路，不仅仅展示了实时设备感知信息，还展示了由实时感知数据净荷经过长期历史数据关联、挖掘后得到的历史目标分析数据。实时态势监控模式包含了如下子功能：

1）多维度数据视图

多维度数据视图是按照目标特征提取、行为分析算法与威胁判别分析模型警行综合态势估计后得到的目标富语义数据，按照数据处理程度由低到高可以分为原始感知数据视图，目标特征数据视图，目标行为数据视图，目标识别和威胁等级数据视图。

原始感知数据视图展现了以目标为中心的各种原始感知数据，展示了关联至同一个目标下的同目标多源异构数据感知数据。以 AIS 信息探测设备与民用雷达进行数据融合为例，原始感知数据视图呈现了经过数据关联、融合后同一个目标的经过 AIS 设备探测到的经纬度信息、船名和经过雷达设备探测到的设备距离、径向速度、方位角等信息。由于原始感知数据量大、超出了传统数据库的存储空间限制，因此，原始感知数据视图可以存储在分布式面向列的数据库中，当感知数据不断累积时，横向扩展存储节点的数量满足其对存储空间的需求。而其索引可以存储在图数据库中，图数据库中存储了目标和各类感知数据索引及它们间的相互关系，体现了以目标为中心的存储理念。

目标特征数据视图呈现了目标本身的属性特征，目标的特征主要分为两类，第一类为信息类，包含了探测设备感知到的目标详细信息数据，例如目标大小、长宽比、颜色、国籍、速度、角速度、电磁波段等信息；第二类为标志类，标志类数据具备唯一性，比如 AIS 信息探测设备探测得到的 mmsi（Maritime Mobile Service Identify 水上移动通信业务标识码）号、航号信息。这种标志是唯一的，可以利用标志特征进行目标多维度信息的关联、融合，实现一个标志盘活一个网络的效果。标志类特征还具备固定性，标志类信息是不易改变的稳态特征，稳定的表征出当前探测目标的状态，如目标的微多普勒特征可以清楚地显示了的结构

部件的时变频率分布。这两类特征数据可以从原始的感知数据中通过特征提取算法计算而来。例如对于目标颜色、长宽比等特征可以通过图像处理中图像增强、分割和二值化等技术进行提取，同时可以结合摄像头的测距功能提取目标的大小，对于目标的速度和角速度等特征，可以基于 AIS 或雷达等手段，采用时空数据处理相关技术进行提取。最终目标特征数据作为实体存储在图数据库中，成为目标特征数据视图[18]。

目标行为数据视图存储的是目标的行为信息，主要包括目标的历史经过区域、篡改或伪装次数等，这些信息可以从原始的感知数据中通过行为分析算法计算出来。例如，对于历史经过区域，可以采用人为划定防护区，再采用船舶进出港相关算法进行统计分析。而对于篡改或伪装次数，可以通过目标信息图谱检索对比进行分析统计，具体而言，当电磁或 AIS 同时感知到目标后，通过电磁手段检索目标信息图谱，可以获取目标的历史 AIS 数据，此时，可以将目标现在的 AIS 信息和历史 AIS 信息进行对比，判断目标是否存在 AIS 信息篡改行为。目标的行为数据随着感知数据的不断积累而更新，这些数据可以帮助我们显式的进行目标威胁分析。最终目标行为数据作为实体存储在图数据库中，成为目标行为数据视图[19]。

目标威胁判据数据视图存储的是目标识别和威胁分析的结果，这些信息是基于以下三层数据（原始感知数据、目标特征数据、目标行为数据），通过显式规则判断或深度学习模型训练预测而来。对于人可理解的威胁判断，可以结合目标行为数据视图根据设定规则进行威胁分析，例如，当目标进入防护区次数超过设定阈值后，就认为目标是威胁的。而对于无法通过规则进行威胁分析的目标，则需要采用深度学习进行威胁分析模型的训练与预测，此时，可以将原始感知数据、目标特征数据、目标行为数据作为输入，将已知的威胁等级为输出，利用深度学习的反馈机制，不断训练，从而得到威胁分析模型，当有新的目标出现时，可以将新目标的各类感知、特征、行为数据作为输入，通过模型对其威胁等级进行预测。

2）目标信息图谱

目标信息图谱展示了系统发现的多种不同的传感手段探测的到的同一个目标的多源异构数据的相关关联关系。比如电磁频谱、雷达、AIS 探测设备、光电视频等设备同时探测到一个目标之后，设备的感知探测数据之间会产生横向、纵向、斜向的关系。同一目标不同设备在相同时间、相同地点的感知数据间产生横向相关关系。同一目标相同设备在不同时间相同地点探测到的感知数据间产生纵向相关关系。同一目标不同时间、不同地点由不同设备探测得到的感知数据产生

斜向关系。目标信息图谱结合时空数据、环境信息、目标信息呈现了同一目标多维度设备数据通过横向、纵向、斜向串联，反复迭代，从点到树，从树到网的目标信息网结构。

3) 实时与历史结合的目标情景回放

目标情景回放分为行为事件情景回放与视频情景回放两部分功能。行为事件情景回放基于实时历史无缝服务切换，实现告警行为与区域行为的数据情景回放。

视频情景回放基于流媒体服务，使用流媒体播放器进行光电设备录制视频情景的回放。

在实时态势监控模式中，目标行为数据视图实时的呈现了目标历史发生的进出防护区行为、告警事件等历史情景，此时可以选中需要进行回放的告警事件或区域行为，通过实时与历史双展示模式无缝服务切换，一键式的从高维度实时态势监控模式的切换至历史回放服务。行为事件情景回放是基于实时与历史结合的思想，是实时模式与历史模式的关联，用历史事件、行为情景辅助实时监控的思想，实现高维度、一键式基于实时历史无缝服务切换的实时态势监控情景回放。

视频情景回放基于后端流媒体服务，在实时态势监控过程中，对于目标光电视频设备历史存储监控信息的回放需求，使用流媒体播放器进行目标历史情景视频的当前界面回放。从视频监控的角度以实时与历史结合的思想，可以在实时监控目标光电视频流信息的同时，进行目标历史视频情景的回放，实现视频角度的实时视频监控与历史情景视频结合的实时态势监控。

4) 设备显控与联动

接入态势呈现系统的的设备的感知数据需要进行实时的可视化，并且用户可以根据当前设备的实时告警、监控信息，对设备进行实时控制命令下发，实现对设备的控制。系统中接入的传感器之间可以具备联动的能力，协同各个设备能力，减少了任务中间环节，提高了效率。

设备显控即对当前设备上报的实时感知动态数据进行动态的可视化，并且还可以在系统浏览器界面对设备进行远程控制，实现控制命令的实时下发。

设备联动通过消息中间件的转发功能，将位置分散、结构各异的传感器有机地结合在一起，使各传感器之间的优势互补，共同完成对目标的精确检测和取证，实现雷达光电联动、AIS 光电联动等设备联动功能。设备联动同时也简化了人工操作的流程，降低了系统的复杂度。

5) 视频监控与跟踪

系统通过接入流媒体服务提供的实时解码视频流，利用多媒体监控手段，采用视频回传快速、功耗略大的光电视频监控设备对目标进行实时监控和跟踪。使

用流媒体播放器在占用系统资源较少的情况下实现高效率的实时视频显示，使用目标跟踪算法与光电设备控制命令下发结合的方式实现光电视频设备单目标实时跟踪。

6) 实时告警与告警跃迁

态势呈现系统的实时告警功能通过多传感器实时感知数据通过一系列多层次的实时告警判别显式规则与异常数据检测算法进行实时目标告警服务，一旦目标出现告警事件，目标渲染标记将会根据目标告警等级实时更新渲染颜色，并且将根据告警程度不不同进行不同频率的闪烁，以智能化、形象化的表征设备警报故障与目标告警信息。

系统的告警跃迁过程实际上是目标威胁判据的计算过程的详细展现，目标的告警等级并不是静态的，并不是根据发生的危险行为、告警事件得到的静态结果。而是发展的、动态的，所有目标在数据刚入库时告警等级均为正常，也就是没有威胁，在长期对其进行监控的过程，中由于目标发生了一些危险行为与告警事件而使其威胁不断提升，比如，目标在某时段作为非协作目标进出了监控预警区，使其威胁等级上升为轻度警告，又在某时段发生了 ais 恶意篡改事件，从而进一步提升了威胁等级，产生了告警跃迁。

历史数据回放模式是对长期实时设备监控积累的目标感知、特征、行为数据进行区域行为、告警事件、全信息、多维度的时间维度历史数据回放。历史回放分为历史回放流程控制、历史回放方式选择、历史回放事件告警三个功能。回放流程控制功能主要包括了缓冲式的批量目标数据加载与速度控制，系统提供了全景、多目标、区域、事件四种历史回放的查询回放方式，并且在回放至包含关键告警事件、区域重要行为信息的时间节点时会自动暂停弹出告警通知，指引进行关键事件信息的停止快进逐帧回放或跳过关键事件信息。以下为历史数据回放子功能的详细介绍：

1) 历史数据回放流程控制

历史数据回放流程控制包括了缓冲式的批量数据加载与速度控制。系统在进行历史回放时候会定时批量加载历史回放缓冲帧数据，系统将一次性缓冲多帧数据，每一帧包括了所有监控目标在当前回放时刻的融合感知数据，这样降低了系统负担，减少冗余网络流量，可以大大增加历史数据回放的流畅性。

历史回放速度由加速、减速按钮控制，当单击加速或者减速按钮后，系统会修改播放倍率，速度因子越大，两帧所记录的融合感知数据之间的时间差越大，实际的回放速度也就越快，从而实现历史回放速度的控制。

2) 历史数据回放方式选择

历史数据回放根据查询条件的需要，提供了四种回放方式。针对全量目标信息、整体态势、大面积全量环境的回放要求，系统提供了全景回放方式，将选定时间段的所有目标的全量信息按照时间维度的流失逐帧全信息回放。针对确定的单目标与多目标详细时段内历史感知信息的回放要求，系统提供目标回放的方式，针对选定的特殊单目标或多目标进行时间维度具体时段内的历史数据回放。针对空间区域划分回放、防护区域历史事件的回放要求，系统提供区域回放方式，针对特定防护区或者鼠标绘制任意矢量区域图形内部的特定时段历史数据进行回放。针对告警事件、异常数据检测的回放要求，系统提供事件回放的方式，针对历史发生的设备报警、目标行为告警、伪装告警等事件进行具体时段的事件回放。

3) 历史数据回放事件告警

快进回放至包含关键告警事件、区域重要行为信息的时间节点时会自动暂停弹出告警通知，选择停止快进逐帧播放还是跳过事件。此时目标渲染标记将会根据目标告警程度更新渲染颜色，并且根据告警程度的大小进行不同定时频率的闪烁，表现出历史回放告警事件与目标告警信息。

目标跟踪判读服务是在空间维度将目标的历史数据用空间大数据可视化的方法进行不同展示方法的空间范式历史空间数据展示回放。实时监控与历史数据回放是基于时间的，局部的，细粒度的展示，而目标跟踪判读是基于空间的、全局范围内的粗粒度的展示。目标跟踪判读的子功能如下：

1）迁徙转移态势

系统将选中的特定单个目标在特定时段内的空间历史轨迹经过数据采样、关键轨迹帧提取、历史轨迹查询后展示出目标的空间迁徙转移态势，并展示出目标迁徙轨迹的目标历史静态时空信息。目标迁徙过程中的发生的事件、区域行为、告警信息，使用高密度、可伸缩的图表渲染在可视化界面之上。

2）热点流动区域

针对单目标经常出现的区域，系统以热力图的形式，通过色带渲染目标空间位置数据疏密程度及区域访问频度，通过核密度分析方法对于单目标在特定时段内出现的每个空间离散位置数据点建立缓冲区，使用渐进的灰度带由内而外，由浅至深的填充，缓冲区交叉的区域，灰度值相互叠加，这块区域也就越热。将叠加后的目标空间位置灰度信息为索引映射至色带中，从而用冷色到暖色展示出目标热力点相对密度与加权密度状态。

3）告警数据蜂窝

系统使用网格聚合图的方式展现出目标告警事件、行为的空间数据的分布特征和统计特征。基于网格聚合算法，将空间区域划分为具备嵌套性的多层次，每

个网格单元都具有目标告警事件的统计信息。

目标画像服务是将目标行为、特征、事件的差异，将他们区分为不同的类型，从每种类型中抽取典型的特征赋予 ID、统计要素等描述，形成一个目标原型，即目标信息标签化，并且对这些特征分析统计挖掘潜在价值信息，建立目标数据立方，抽象出一个目标的特征全貌。目标画像能够帮助安防监控者找到对的真正需要关注的目标，分析统计挖掘潜在价值信息从而能够一定程度上预测这个目标下一步的行动，目标画像包含如下子功能：

1）目标信息标签化

系统通过一系列算法或规则挖掘得到或者根据行为数据直接得到目标特征信息，并且将其标签化，特征是有一定的时效性或者半衰期的，针对不同时间得到的标签需要进行权重加权处理。目标的感知特征标签包括国籍、长宽、船类型、目的地、旋转速率、径向速度、电磁频率、实际航向等，目标行为属性标签包括进出防护区情况、告警情况、出没区域情况（以经纬度形式呈现）、船只状态事件类型（地区问题、人员落水、速度异常、转向过大、持续靠近岛屿、尺寸异常、状态异常、船名异常）、防护区停留时间、靠近岛屿最近距离等。

2）目标原型数据立方

目标画像不仅仅是将目标的行为特征抽取出来赋予目标一个类型或者原型，它还能展示这个原型本身的信息，即我们建立的情报库中关于这一类的目标的一些历史行为、威胁告警数据，这就实现了对这些特征建立数据立方，分析统计挖掘目标原型的潜在价值信息从而能够一定程度上预测这个目标下一步的行动。假设我们实现了电磁管控，某一个目标被打上了超高频通信的标签，但是它没有触发任何的告警事件，也暂时没有产生危险行为，但是这一类高频通信的历史目标中大部分都是威胁等级高的目标，可以对当前目标做一定程度的预测。

## 3.3 系统服务结构设计

系统采用实时与历史结合的面向服务的架构，从服务交互的角度来看整个系统，实时服务与历史服务之间互相作为服务生产者与消费者。具体来讲，实时监控模式的多角度数据视图服务、目标信息图谱服务是历史挖掘服务的服务消费者，历史挖掘服务提供了特征、行为、威胁分析数据，是服务的生产者；在历史数据回放服务是实时接入服务的服务消费者，物联网实时接入服务是服务数据生产者。

系统的服务具备内聚性，根据实时监控与历史回放业务的不同，确定了服务边界，使用组件化这种抽象层保证相关的业务代码放在一起，实时与历史服务之

间有确定的边界，具备一定弹性，如果系统中的一个实时或者历史服务组件不可用了，但是没有导致级联故障，那么系统的其他部分组件还可以正常运行，服务组件保证了服务边界的存在。

实时服务与历史服务具备自治性，实时历史服务之间通过网络调用进行通信，加强了服务之间的隔离性，避免紧耦合。实时与历史服务都会暴露出 API（Application Programming Interface 应用编程接口）,服务之间通过 API 进行通信，修改一个服务并对其进行部署不影响其他服务。

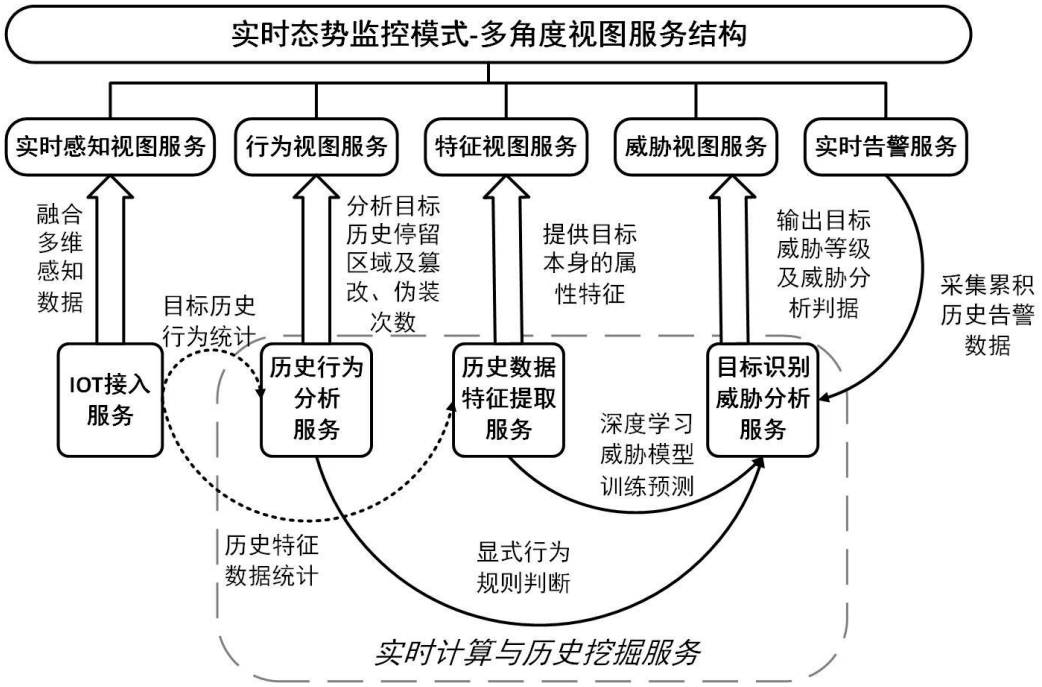


**图 3-8 实时监控模式多角度视图服务结构**

实时态势监控模式多角度视图服务由感知数据视图、行为感知数据视图、特征感知数据视图、威胁感知数据视图、实时告警服务聚合而成。从服务交互的角度看，它们是 IOT 接入服务与实时计算与历史挖掘服务的消费者。IOT 接入服务生产出了目标原始感知数据，这些数据由实时计算与历史挖掘服务通过特征提取、显式规则判别、威胁模型训练等方式所消费并又二次生产出目标行为、特征、威胁数据视图，最终这些二次生产数据被多角度视图服务所消费，展示出目标的多维度信息视图。实时监控模式多角度视图服务结构如图 3-8 所示。

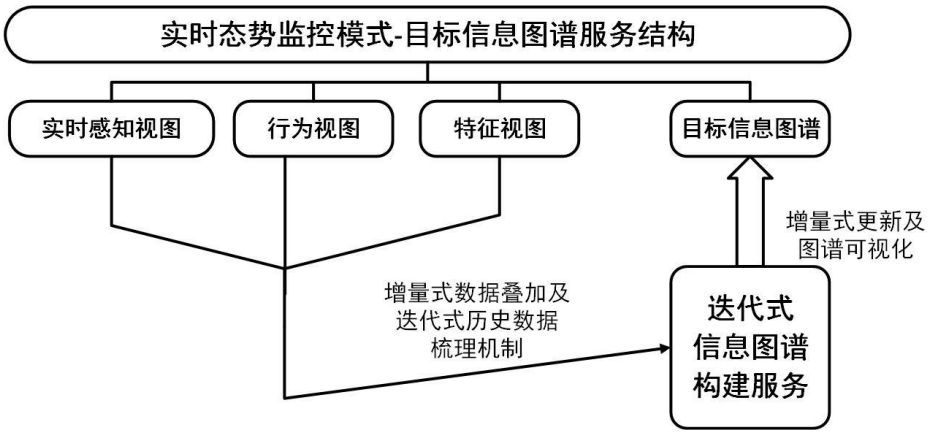


**图 3-9 实时监控模式目标信息图谱服务结构**

实时态势监控模式目标信息图谱服务目标信息图谱展示了系统发现的多种不同的传感手段探测的到的同一个目标的多源异构数据的横向、纵向、斜向的相关关联关系。当各类感知数据累积到一定阈值后，就会触发增量式的数据叠加和历史数据梳理机制，迭代式信息图谱构建服务不断的消费感知、行为、特征视图数据，从而不断的更新目标信息图谱中的特征和行为信息，不断整合、删除、更新各个时段的图谱片段。目标信息图谱结合时空数据、环境信息、目标信息呈现了同一目标多维度设备数据通过横向、纵向、斜向串联，反复迭代，从点到树，从树到网的目标信息网结构。实时监控模式目标信息图谱服务结构如图3-9所示。

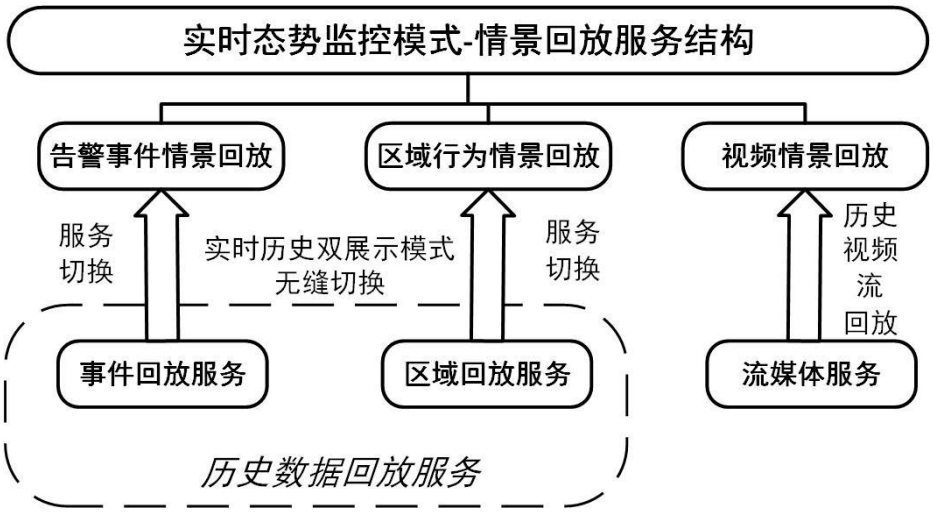


**图 3-10 实时监控模式情景回放服务结构**

实时态势监控模式情景回放服务由告警事件情景回放、区域行为情景回放、视频情景回放服务聚合而成。通过实时与历史双展示模式无缝服务切换，一键式的从高维度实时态势监控模式情景回放服务的切换至历史回放服务。在这里实时

监控模式情景回放事件行为情景回放服务是服务消费者，历史数据回放服务是服务数据生产者。这种一键式实时历史服务切换方法体现了实时模式与历史模式的关联，用历史事件、行为情景辅助实时监控的实时历史结合的服务交互的思想。视频情景回放基于后端流媒体服务，从视频监控的角度以实时与历史结合的思想，可以在实时监控目标光电视频流信息的同时，进行目标历史视频情景的回放，实现视频角度的实时视频监控与历史情景视频结合的实时态势监控。实时监控模式情景回放服务结构如图 3-10 所示。

实时态势监控模式设备服务由设备显控、设备联动、视频监控、目标跟踪四个服务聚合而成。物联网接入服务为设备服务暴露了上行数据报文与控制命令下发的应用程序接口，流媒体服务为设备服务暴露了视频流接入及光电设备控制的应用程序接口，最终实现多传感器之设备间的协同联动来综合多方面的数据，完成对目标的设备检测与监控。实时监控模式设备服务结构如图 3-11 所示。

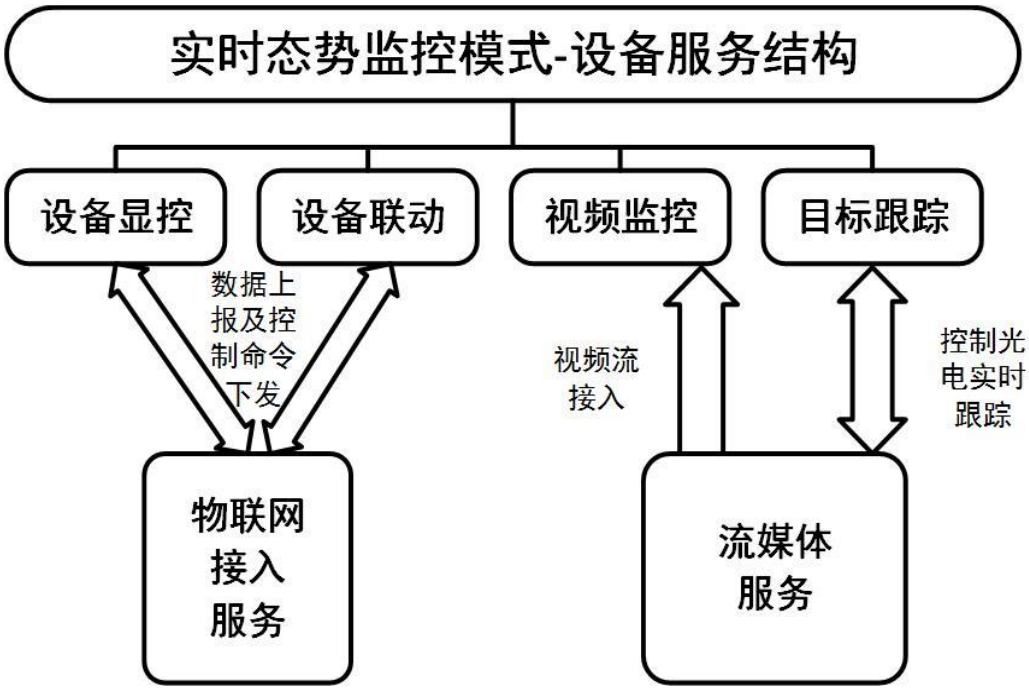


**图 3-11 实时监控模式设备服务结构**

历史数据回放服务包括了事件回放、区域回放、目标回放、全景回放四种服务方式，历史数据回放服务通过缓冲式的批量数据加载对历史数据存储服务的数据进行消费，历史回放服务会定时批量加载历史回放缓冲帧数据，即一次性缓冲包括了所有监控目标在当前回放时刻的融合感知数据的批量帧数据，这样增加历史数据回放的流畅性，并且降低了冗余网络流量，减少系统负担。

历史数据回放服务的原材料由历史数据存储服务提供，其数据是经过物联

网实时接入服务存储至实时数据存储服务中，再经过实时计算与历史挖掘服务实现目标维度与区域维度的数据关联、融合，并进行特征行为分析与威胁分析模型训练，将数据关联、融合、分析、挖掘的结果存储到历史数据存储服务中，用来为历史数据回放服务提供缓冲式的批量时空历史数据加载功能。历史回放服务结构如图 3-12 所示。

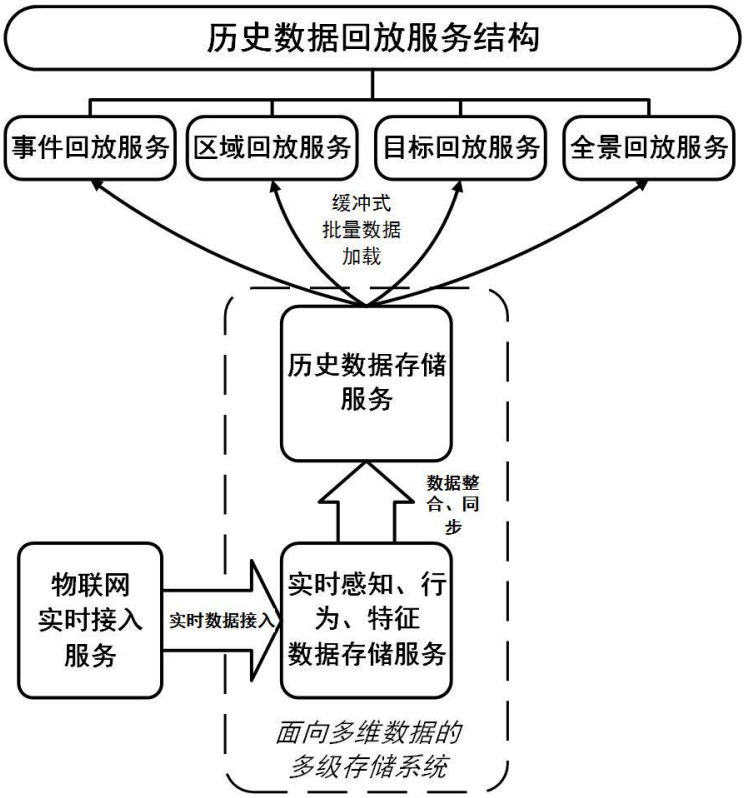


图 3-12 历史回放服务结构

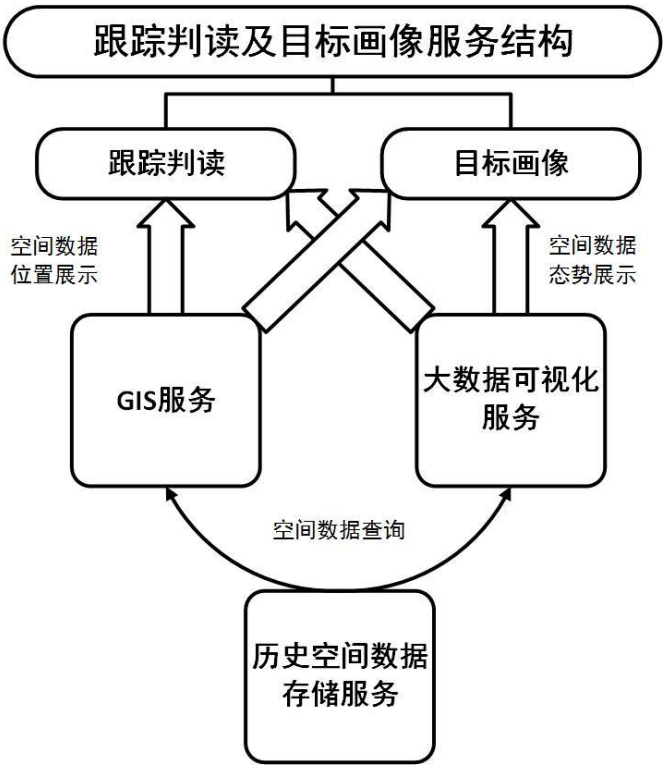


**图 3-13 跟踪判读及目标画像服务结构**

跟踪判读服务标跟踪判读服务是在空间维度将目标的历史数据用空间大数据可视化服务进行空间数据可视化。目标标跟踪判读具备空间性、全局性与粗粒度的特征，它是历史空间数据存储服务的服务数据消费者，将空间历史轨迹、单目标区域访问频度、区域目标告警事件分别使用迁徙转移图、热力图、网格聚合图的方法展示在地图上。跟踪判读及目标画像服务结构如图 3-13 所示。

## 3.4 系统组件化结构设计

态势呈现系统使用组件化架构将多个功能模块拆分、重组，从而分离组件边界和责任。态势呈现系统将需求场景化、视觉表达组件化，实现组件隔离、组件复用、组件自治，从而将复杂系统分拆成为微型组件元素进行部署维护。组件有多种属性，其状态反映内部特性。组件化具备标准性，系统中的组件体系都应该符合统一的设计标准，形成一套统一化的套件库。组件化具备组合性，组件之间通过一定编排组合后可以提供针对性的业务服务。这种服务依赖不同组件之间的通信、组装与嵌套。每一个组件都的功能都具备独立性，不依赖于其他组件，并且自身逻辑与其他组件无关。组件化增加了系统复用性，灵活性，提高系统设计，从而提高开发效率。

系统采用自顶向下的嵌套式组件化设计模式，将整个实时历史双展示模式的态势呈现系统对外表现成为一个整体系统组件。根据区域特性将整体系统组件进行划分后，整体系统组件内部由控制区组件与视图区组件组合而成。组件区域划分如图 3-14 所示。

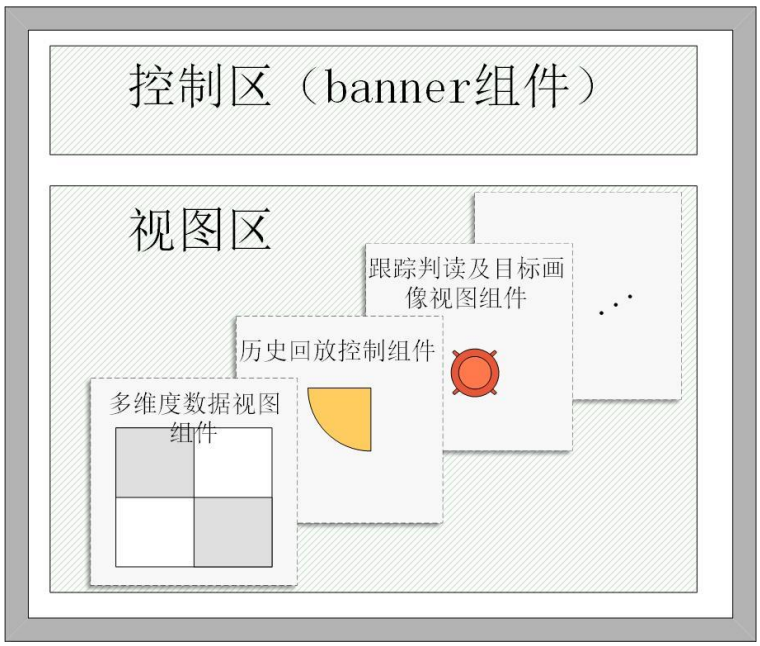


**图 3-14 组件区域划分图**

控制区组件主要负责系统服务核心控制与交互功能，负责实时态势监控模式、历史数据回放模式、跟踪判读与目标画像的服务启动，服务终止，服务切换功能，由于采用了组件化的设计架构，各个服务之间可以独立运行，不会相互影响，例如系统可以在进行实时目标态势监控的同时，对当前目标的历史行为历史事件进行回放，两种服务之间具备隔离性，不会相互影响。故控制区组件具备高内聚、低耦合的特性。

视图区组件负责进行系统所有服务的可视化任务。比如实时态势监控模式中目标标记节点的呈现、目标标记告警闪烁渲染、跟踪判读服务中的热力图、迁徙图、蜂窝图的绘制及地图渲染、多维度数据视图及目标信息图谱的绘制等。并且视图区组件可以进行可视化的交互，例如历史数据回放模式中的速度控制、告警事件逐帧数据回放交互等。

系统整体详细组件化架构如图 3-15 所示。

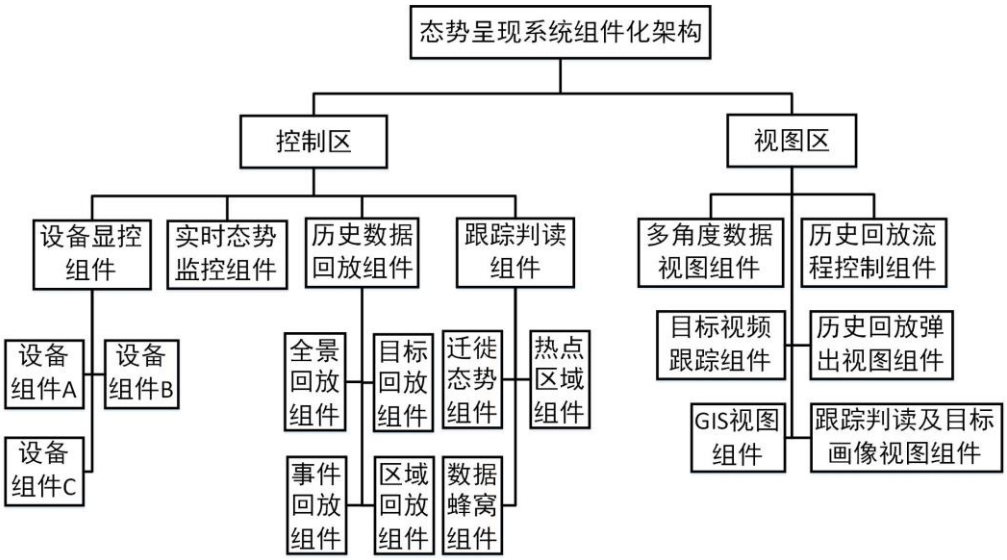


**图 3-15 系统组件化架构图**

态势呈现系统组件化架构分为两个区域组件，控制区域组件与视图区域组件，控制区域组件由设备显控组件、实时态势监控组件、历史数据回放组件、跟踪判读组件四个组件组合而成。视图区组件由多角度视图组件、历史回放流程控制组件、历史回放弹出视图组件、目标视频跟踪组件、GIS 视图组件、跟踪判读及目标画像视图组件组成。

控制区域组件中的设备显控组件包含了所有系统中的探测设备组件，它们负责每个设备服务的启动、终止功能。实时态势监控组件使用鼠标事件控制系统进入实时态势监控模式。实时监控模式服务启动后，通过独立组件通信机制通知视图区中的多角度数据组件启动多维度数据视图服务。历史数据回放组件由全景回放组件、目标回放组件、事件回放组件、区域回放组件组合而成，每个组件代表了一种历史数据回放服务的启动方式，按照特定方式进行历史回放服务启动包括了批量数据索引预加载、视图区组件通信、视图区组件渲染等流程。历史回放服务启动后，通过信号广播器通知历史回放流程组件与历史回放弹出视图组件进行预渲染，启动历史回放服务批量数据缓冲可视化流程。跟踪判读组件控制了跟踪判读服务中的迁徙态势、热点区域、数据蜂窝的服务启动、终止。通过鼠标事件启动跟踪判读服务后，通过组件化通信机制，通知视图区 GIS 视图组件进行大数据可视化渲染迁徙图、热力图、网格聚合图。

视图区组件中的多角度数据视图组件由目标信息图谱组件、目标特征数据视图组件、目标行为事件数据视图组件、目标威胁分析组件组成，主要展示通过鼠标事件选择的特定目标的多角度视图的目标特征行为、威胁分析，感知数据，并

且可以进行一键式的情景回放，实现实时历史服务无缝切换。历史回放流程控制组件主要提供缓冲式的批量数据加载可视化及回放流程、速度控制服务。历史回放弹出视图组件提供历史数据回放事件告警服务，快进回放至包含关键告警事件、区域重要行为信息的时间节点时会自动暂停弹出告警通知，此时目标渲染标记将会根据目标告警程度更新渲染颜色，并且根据告警程度的大小进行不同定时频率的闪烁，表现出历史回放告警事件与目标告警信息。跟踪判读及目标画像视图组件主要进行相应的视图渲染任务，GIS 地图组件负责大数据可视化与瓦片地图渲染服务。

## 3.5 系统数据流设计

上行数据流是原始感知数据从物联网设备中生产出来，依次通过 IOT 接入服务、消息中间件、关系数据库、实时计算与历史挖掘服务、列式与图数据库和态势呈现系统的模型、控制器、视图层的组件渲染与状态转移，最终实现数据可视化的流程。系统上行数据流程图如图 3-16 所示。

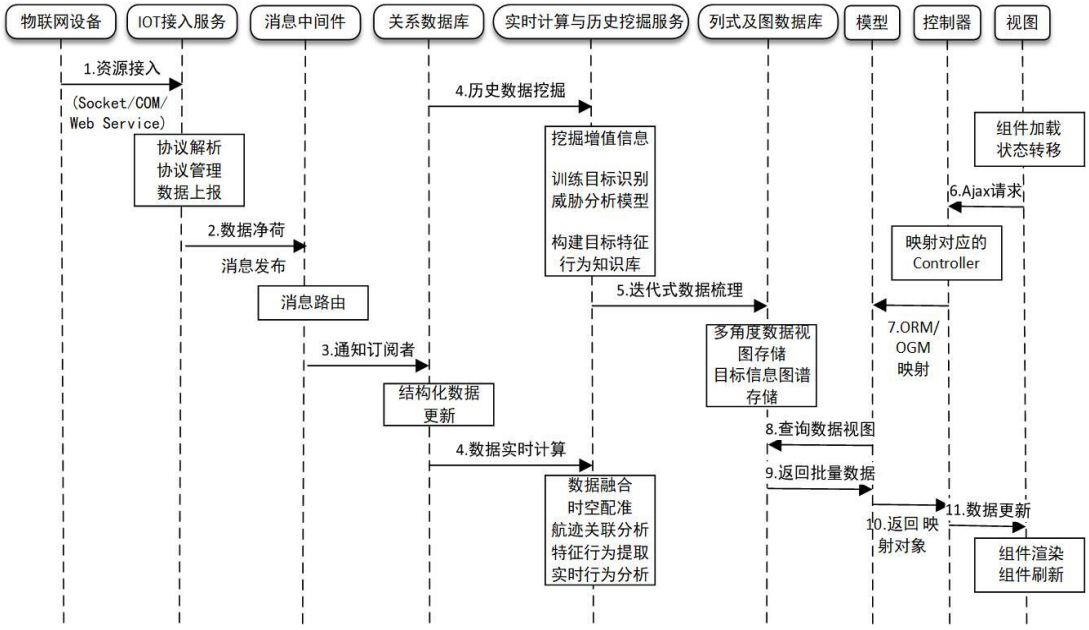


**图 3-16 系统上行数据流程图**

1）多源异构的物联网传感器设备资源通过 Socket、COM 或者 Web Service 等通信方式将物联网传感器原始数据报文接入到物联网接入服务中。

2）物联网接入服务将原始设备生产出原始感知信息，进行协议解析、协议

管理、数据净荷提取、报文分发等过程将数据报文以一定主题发布至消息中间件。

3）消息中间件采集发布至消息空间的相应消息树后，将消息缓存至消息队列中，消息队列中缓存的消息经过消息路由发布至订阅了该条消息的关系数据库中，关系数据库进行结构化数据实时更新。

4）实时计算与历史挖掘服务的实时计算部分将实时的原始感知数据进行时空配准、轨迹关联分析、特征行为提取、实时行为分析、数据融合后，发现多种不同的传感手段探测到的同一个目标的多源异构数据之间的相关关联关系，并将同一目标的多种手段不同维度的感知数据关联至同一个目标下。历史挖掘部分将目标长期运动轨迹、历史碎片化数据进行并行化的离线挖掘分析，通过特征提取及行为分析算法得到目标本身的属性特征信息与目标行为信息，并且训练目标识别威胁分析模型，挖掘目标富语义增值信息，构建目标行为知识库。

5）将实时计算与历史挖掘服务形成的目标行为知识库进行反复迭代、增量计算、数据累积后形成含有目标感知、特征、行为、威胁数据视图的目标信息图谱，并将多角度视图与目标信息图谱分别存储至列式及图数据库中。

6）态势呈现系统组件加载、组件状态转移后，发起 Ajax 异步数据通信，请求批量数据视图。

7）控制器经过数据预加载等计算后使用 ORM/OGM 映射的方法获取批量数据对象。

8）在图数据库与列式数据库中使用数据库语句查询多维度数据视图与目标信息图谱中的节点、关系信息。

9）从图数据库与列式数据库中返回批量数据被态势系统模型层捕获。

10）从模型层返回映射为普通 Java 对象的批量视图数据。

11）POJO 对象被控制器解析为 JSON 格式，并且进行自适配的视图组件更新、视图组件渲染。

下行数据流从用户进行界面控制开始，经过视图层、控制器、消息中间件、IOT 接入服务，最终控制命令下发至物联网设备中，实现设备显控与联动、目标跟踪与设备告警。系统下行数据流程图如图 3-17 所示。

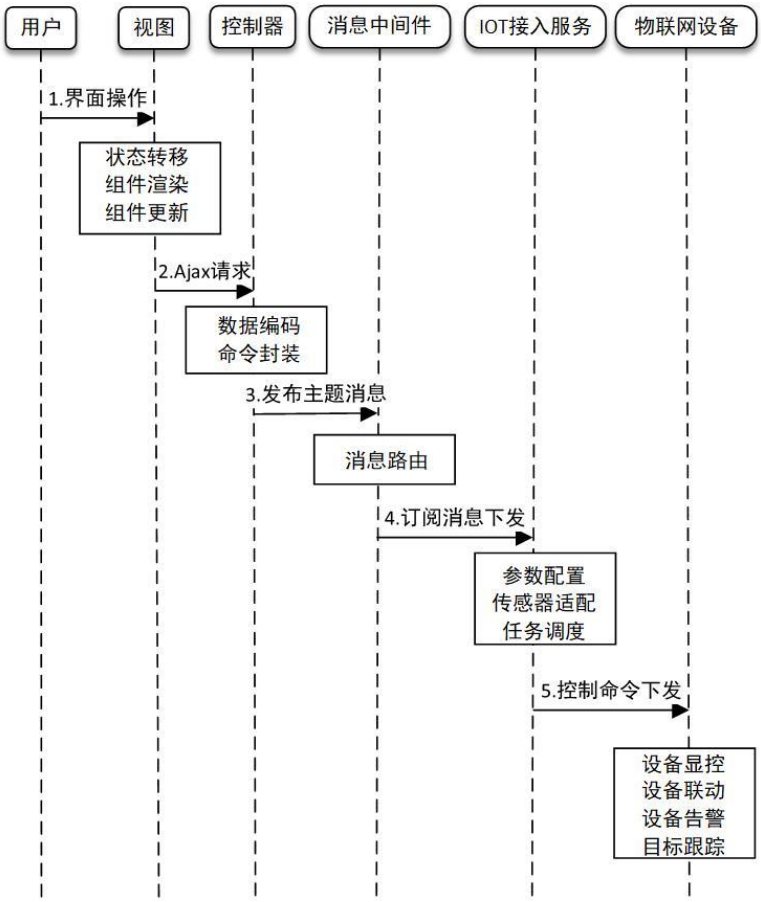


**图 3-17 系统下行数据流程图**

1）用户通过操作鼠标事件，使视图组件进行状态转移、组件渲染、组件更新。

2）视图层经过状态转移、组件渲染、组件更新后使用 Ajax 发起异步通信，将控制命令报文通过异步通信机制传递给控制器。

3）控制器将控制命令进行数据编码、命令按照约定协议封装后。将控制命令报文按照特定主题发布给消息中间件

4）消息中间件接受到消息后将消息发布至缓冲队列中，然后通知订阅了该主题的物联网接入服务。

5）物联网接入服务经过协议栈参数配置、传感器适配、下发任务调度后，将该控制命令下发给相应设备，实现设备显控与联动、目标跟踪与实时告警。

## 3.6 系统总体流程设计

实时与历史结合的多视图态势呈现系统中分为三类流程操作，实时态势监控

流程操作、历史数据回放流程操作与跟踪判读与目标画像流程操作。

实时态势监控流程如图 3-18 所示。

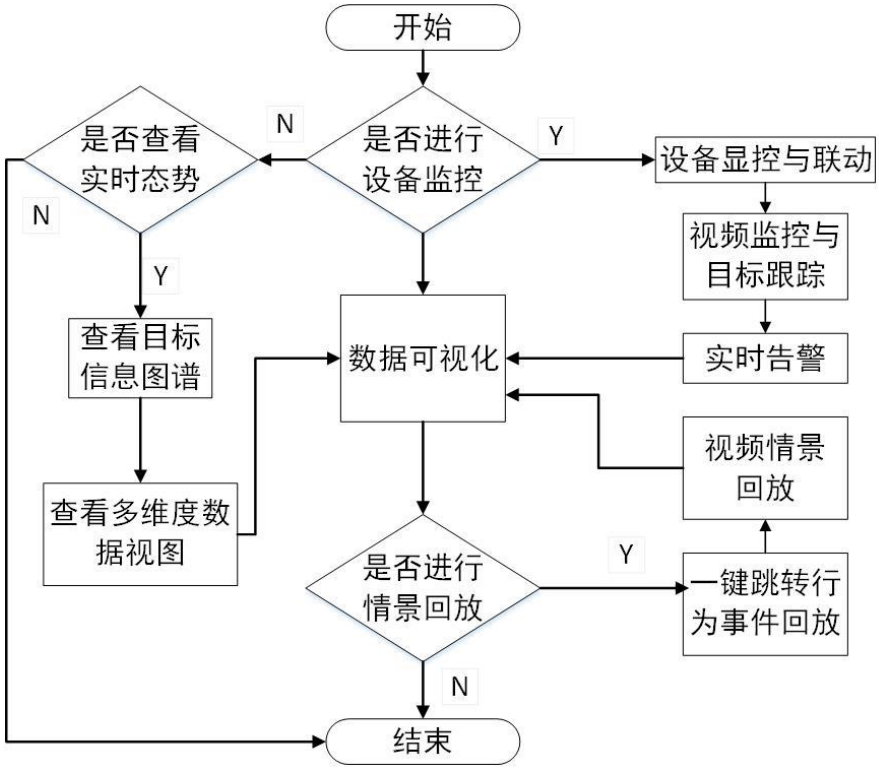


**图 3-18 系统实时态势监控流程图**

用户通过鼠标事件启动实时态势监控模式后选择是否进行设备显控，若进行设备显控，则选择一个具体设备进行设备显控与联动功能，再通过设备联动调用光电视频监控与目标跟踪功能，最终将目标静态、动态、实时告警数据进行可视化。若不进行设备显控，用户需要选择是否开启实时态势监控模式，开始实时态势监控后首先查看目标信息图谱，然后查看多维度数据视图，最终将目标多视图数据进行可视化，查看目标行为事件数据视图后需要选择是否进行情景回放，首先选择具体需要进行情景回放的事件与行为，然后一键跳转进行实时历史服务切换，打开情景回放控制界面，选择视频情景回放，最终进行数据可视化。

系统历史数据回放流程图如图 3-19 所示。

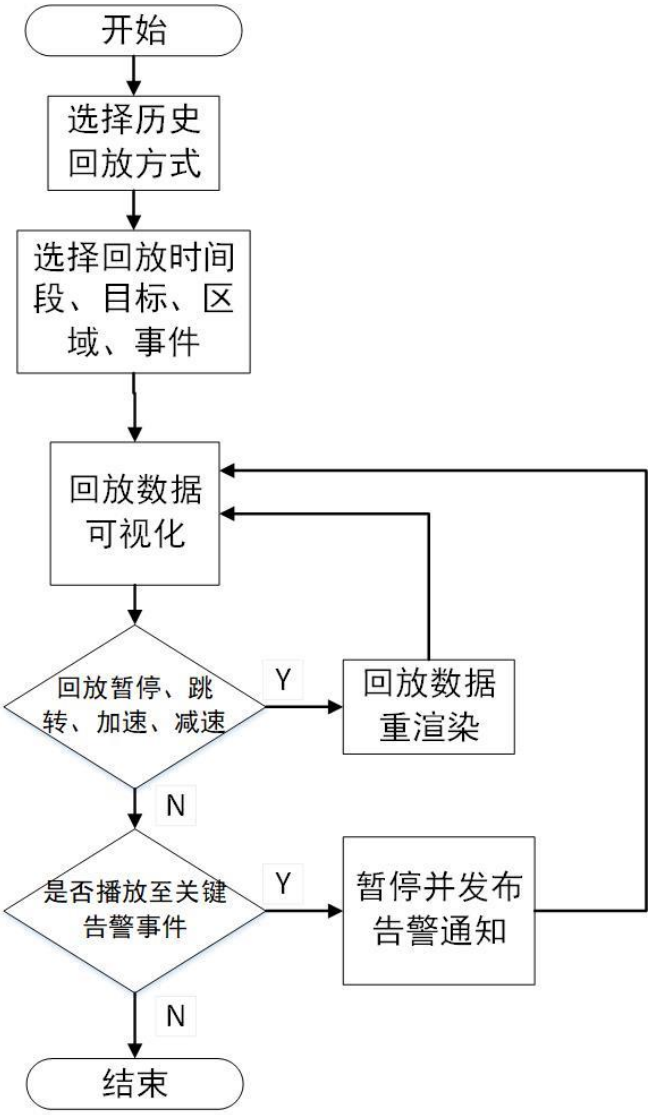


**图 3-19 系统历史数据回放流程图**

系统开始历史数据回放模式后首先根据具体的历史回放需求选择全景、目标、区域、事件其中一个历史数据回放方式，然后根据特定的回放方式选择回放的时间段、目标、区域、事件，接着打开历史回放控制界面进行回放数据可视化，回放的过程中若通过鼠标事件拖动进度条或者单击暂停、加速、减速按钮对回放进行流程控制，则会在相应控制时间点进行回放数据重新渲染，若回放至包含关键告警事件、区域重要行为信息的时间节点时会自动暂停弹出告警通知，此时目标渲染标记将会根据目标告警程度更新渲染颜色，并且根据告警程度的大小进行不同定时频率的闪烁，表现出历史回放告警事件与目标告警信息。

系统跟踪判读与目标画像流程图如图 3-20 所示。

首先选择是否进行跟踪判读服务，若进行目标跟踪判读，则需要选择特定目

标和特定的时段，系统将该目标在该时间段内的空间数据采样，然后依次选择展示目标迁徙态势、目标热力区域、目标数据蜂窝。若进行目标画像，需要使用鼠标事件选择特定目标，然后查看目标特征分布图与目标数据原型立方可视化效果。

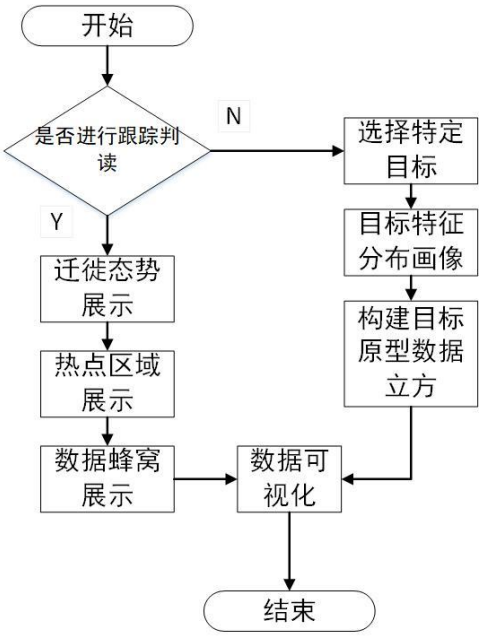


**图 3-20 系统跟踪判读与目标画像流程图**

## 3.7 本章小结

本章详细说明了实时与历史结合的多视图态势系统的总体设计方案。首先，分析了态势呈现系统的整体架构，对系统架构进行了概述，然后介绍了面向多维度的多级存储系统，接着进行接口设计与组件化通信机制的概述。随后进行了系统功能结构的设计，从功能的角度对于系统整体进行了划分。然后阐述了系统的服务结构，从服务交互的角度，阐明了实时与历史结合的设计思想。然后使用自顶向下的组件化架构对系统整体进行了设计，说明了组件化的结构以及各个组件的具体功能，接着说明了系统的上下行数据流，最后对于系统整体的流程做了概述。

# 第四章 态势呈现系统的时空范式设计与组件化实现

在 3.2 节中，本文介绍了态势呈现系统的功能结构，态势系统按功能模块划分见图 3-7，而本系统在 3.7 小节中阐述了可视化界面采用组件化架构实现，系统的组件化架构见图 3-15。系统采用时间范式设计了实时态势监控模式与历史数据回放模式，采用空间范式设计了目标跟踪判读与目标画像服务，下面将结合系统的时空范式设计方案与组件化实现方法，依据系统的功能结构与组件化结构逐个介绍系统每个功能的设计方法与组件化实现方案。

## 4.1 态势呈现系统时间范式设计与组件化实现

态势呈现系统按照时间范式的设计方案分为实时态势监控模式与历史数据回放模式。实时态势监控与历史数据回放都是是基于时间维度，随着时间的流逝将实时或历史的综合态势用多维度数据视图的方法展示出来。基于时间范式的态势呈现强调局部化、细节化的，针对每一个细分的时间片段可以做到全方位、无死角、细粒度的展示。从时间的角度对综合态势实现快速与准确的认知。

### 4.1.1 可无缝切换的双展示模式设计与组件化实现

实时与历史结合的双展示模式采用组件化架构，组件具备内聚性，使用组件化这种抽象层保证相关的业务代码放在一起。实时与历史服务之间有确定的边界，具备一定弹性，如果系统中的一个实时或者历史服务组件不可用了，但是没有导致级联故障，那么系统的其他部分组件还可以正常运行，服务组件保证了服务边界的存在。

双展示模式之间具备隔离性，实时监控模式与历史回放模式之间具备低耦合的特点，两个模式之间不会互相影响，既可以分别展示两个模式的态势信息，也可以在进行实时目标态势监控的同时，对当前目标的历史行为历史事件进行回放。

双展示模式服务激活区域设置在系统控制区组件中，通过下拉菜单组件与鼠标事件实现双展示模式的启动、切换、终止功能。实现双展示模式的控制组件结构如图 4-1 所示。

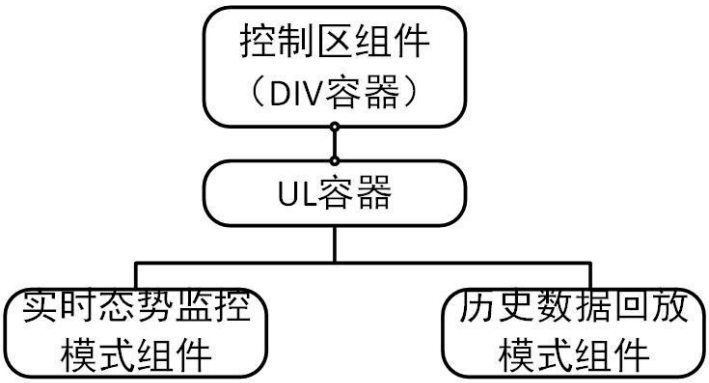


**图 4-1 无缝切换的双展示模式组件结构图**

控制区组件主要负责系统服务核心控制与交互功能，负责实时态势监控模式、历史数据回放模式的服务启动，服务终止，服务切换功能。控制区组件实际上是一个 DIV 容器。DIV 容器中的子元素通过设置 CSS 属性的 float 赋值实现横向排列布局。当元素的布局模式设置为 float 值后，元素将会自动向左或者向右根据容器的规模进行浮动。若容器的规模比较大，元素将不断的浮动至下一横行，直到所在横行的容器规模具备足够空间为止。

控制区组件的下拉功能可以通过鼠标事件进行激活。下拉弹出组件纵向集合菜单功能通过 CSS 的伪类属性:hover 实现。DIV 容器中的元素产生鼠标悬停状态激活事件后，元素产生状态转移，转移函数将元素的:hover 伪类属性激活，将会从鼠标指针区域元素之下将弹出纵向组件集合菜单。当鼠标停止悬停状态离开元素后，元素将不再具备伪类属性，即:hover 伪类属性被取消，下拉纵向组件集合菜单消失。

控制区组件本身并不具备数据可视化的功能，它是一个容器，容器的内部组件负责具体的实时态势监控模式与历史数据回放模式服务启动、服务切换、服务终止的功能，以及使用组件信号广播器向其他视图区独立组件发起通信，实现信息交互。控制区组件进行实时历史双展示模式切换的流程如图 4-2 所示。

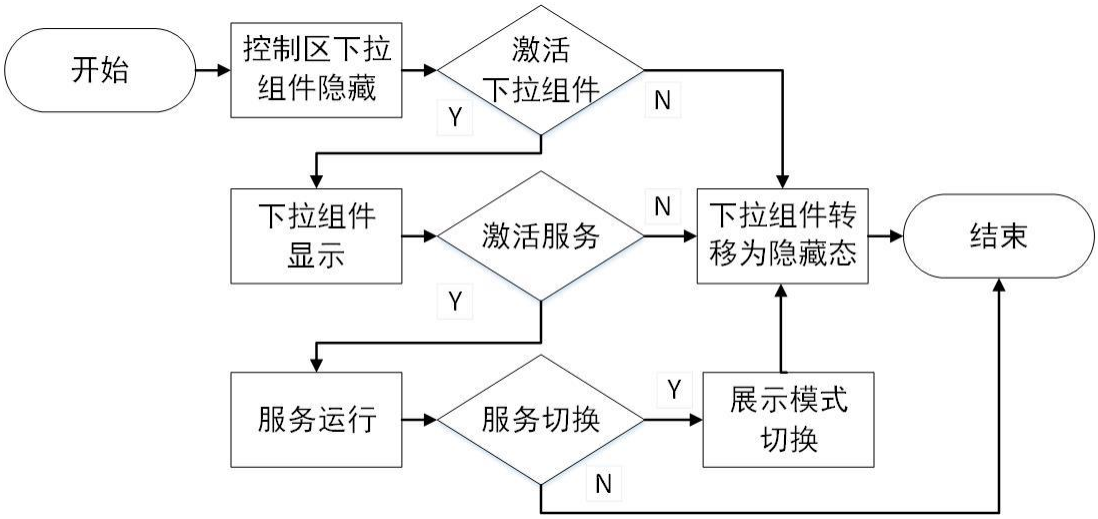


**图 4-2 双展示模式切换流程图**

控制区下拉组件菜单默认为隐藏状态，当鼠标产生移动激活时间后，组件状态改变，状态转移函数向 DIV 容器中元素激活其:hover 伪类属性，渲染纵向下拉组件菜单，用户使用下拉组件菜单进行服务激活，开始进行实时态势监控模式或者历史回放模式，当服务开始运行后，用户进行服务切换，继续在下拉组件菜单中激活或者终止服务，可以实现实时历史双展示模式的无缝、低耦合服务切换。且两个模式之间不会相互影响，可以在进行实时目标态势监控的同时，对当前目标的历史行为历史事件进行回放。

### 4.1.2 实时态势监控模式设计与组件化实现

#### 4.1.2.1 多维度数据视图组件实现

多维度数据视图组件是一个树状嵌套组件集。它可以展示可以存储在分布式面向列的数据库中的以目标为中心的各种原始感知数据。多维度数据视图组件按照数据处理程度由低到高，具备展示原始感知数据视图，目标特征数据视图，目标行为数据视图，目标识别和威胁等级数据视图等多视图的功能。多维度数据视图组件使用 React 框架遵循 ECMA（European Computer Manufactures Association）6.0 标准构建而成。组件最外层是 DIV 容器，DIV 容器内装载了一个使用 ant-design UI 设计语言构建的 Card 组件，Card 组件中的内容部分嵌套了 Tabs 组件，Tabs 组件包含了四个数据视图子组件。因此多维度数据视图组件是一个五层继承嵌套的树形组件。多维度数据视图组件结构如图 4-3 所示。

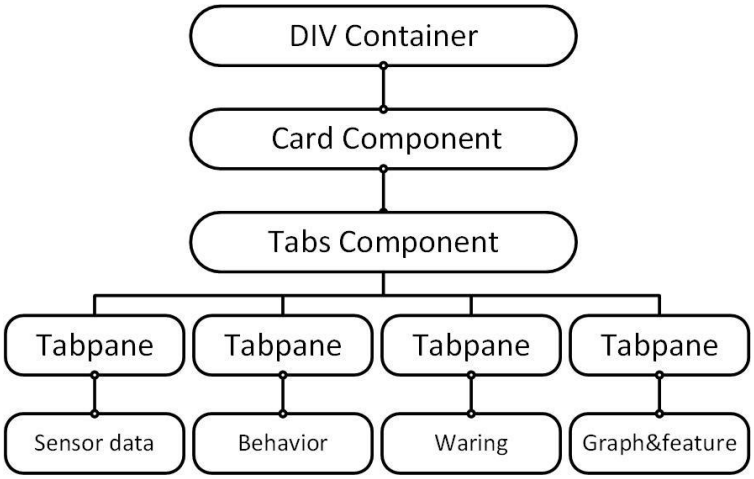


**图 4-3 多维度数据视图组件结构图**

多维度数据视图中的 Card 组件是一个基础卡片容器，它承载了呈现感知、行为事件、特征、威胁数据的文字、列表、图形、段落形式，构建起了一个概览视图骨架。在 Card 组件内部使用 Tabs 组件为感知、特征、行为、威胁视图提供平级的大块内容收纳展现区域，保证了视图的整洁化、面板化。Tabs 组件作为卡片式的页签，在容器顶部提供了可关闭的功能并且关闭函数可以进行重写，实现与其他组件之间的信号交互，并且其具备标准线条式视图切换的功能，可以使得感知、行为事件、特征、威胁进行流畅、平滑、线条式的切换。Tabpane 作为内容面板组件，将具体的多数据视图子组件进行容纳。

多维度数据视图组件使用 activekey 作为当前激活 tab 组件的键值，该键值存储了当前激活的 tab 组件的状态，如果当前激活的视图组件改变，用户进行了组件切换，比如从原始感知视图组件渲染激活状态更改为行为事件数据视图组件渲染激活态，此时多维度数据视图组件通过重写 React 框架中的 componentDidUpdate 方法与 componentWillupdat 方法结合 Tabs 组件的回调函数 onchange 的方式进行激活视图重新渲染，比如组件从威胁判据数据视图经过切换面板动画线性切换至原始感知数据视图后，将重新获取最新的目标原始感知数据，保证多维度数据视图组件面板上展示数据的实时性。在本章接下来的几个小节内将具体展开介绍每个多维度数据视图内部子视图组件的实现方式。

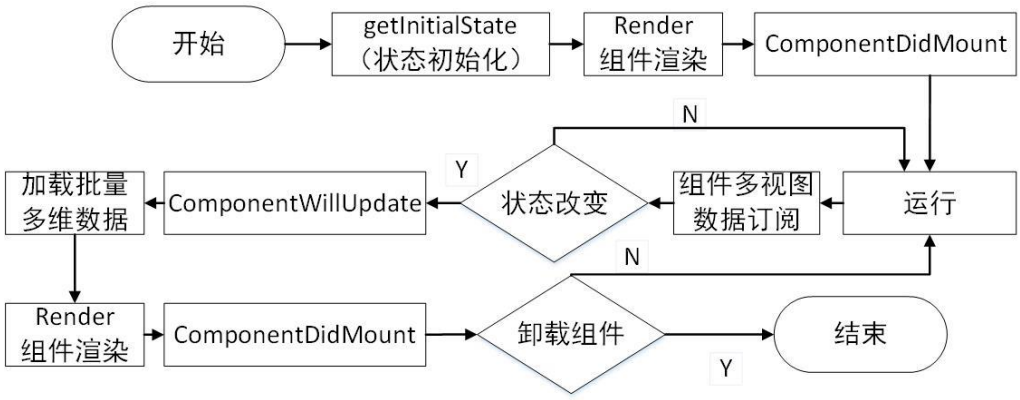


**图 4-4 多维度数据视图组件生命周期**

多维度数据视图组件中的感知、特征、行为、事件数据流遵循了图 3-16 所示的系统数据流模式，其从整体上看，是从数据的生产源头物联网设备经过接入、消息路由、数据关联、数据挖掘、对象映射、前后端通信等流程一步步的进入多维度视图组件中的。

从组件局部的角度上来看，多维度的感知、特征、行为、事件、威胁数据数据是通过图 3-5 所示的组件化独立组件通信机制，实时的订阅了定时器使用周期性的请求的数据，当用户产生了鼠标事件，单击了由定时器控制的周期实时渲染的目标标记后，此时目标标记将会向信号广播器发送信号，同时向服务端进行通信，查询数据库内存储的该目标多维度信息，并且多维度数据视图通过订阅信号广播器多维度数据主题的信号，获取到当前发生鼠标事件的目标信息的多维度数据流，并通过 React 框架内 JSX 语言的自适配渲染语法，将数据按照内部子视图组件的不同，自适配的进行分发，即实现感知数据自适应匹配流向原始感知数据视图组件，行为事件数据自适应匹配的流向行为事件数据视图等。

数据流向子数据视图后，子数据视图组件状态发生改变，将会按照视图组件的生命周期运行 ComponentWillUpdate 方法，首先将批量的多维度数据自适配的流入虚拟 DOM 组件中，通过 React Virtual DOM 组件渲染更新算法计算出最高效率、少步骤的最快渲染方式后，使用 Render 函数对于真实 DOM 进行渲染，并且运行 ComponentDidMount 方法进行组件视图的最后更新，这样就完成了多维度数据从源头到组件，从组件状态改变到组件加载视图，最终到组件渲染更新完成的一系列流程。多维度数据视图组件的整体生命周期如图 4-4 所示。

视图组件静态数据加载是从鼠标事件单击目标节点开始，此时目标标记发送会向信号广播器发送信号，同时向服务端进行通信，查询数据库内存储的该目标多维度信息，组件通过订阅信号广播器的信号直接可以得到包括了目标经纬度、

方位、径向距离、电磁频段、mmsi 号等原始感知的信息，用于渲染原始感知数据视图。同时，定时器控制下的目标标记向服务端通信请求的目标行为、事件、特征、威胁等信息将会通过 ajax 的回调函数发送信号给多维度数据视图组件，使其获得完整的多维度数据。多维度数据视图中的静态数据加载算法如图 4-5 所示。

**Algorithm 1** Heterogeneous Dimension Static Data Loading

**Input:**
The Identification of current selected target $I$.
The State of current selected target state change $S$.
**Output:**
The matrix Sensor data $D$.
The matrix Feature data $F$.
The matrix Behavior and Event data $B$.
The matrix Warning data $W$.
The matrix Graph data $G$.

```
Create a heterogeneous dimension Static Data heap H;
while (S == True) do
    Subscribe sensor data messagge M := getSubscribeSensorMessage(H);
    D := getSensorDataMatrix(H, M);
    Create a connection C to database;
    if (getMulti − dimensionDataElement(H, C)! = null) then
        F := getFeatureDataMatrix(H, C);
        B := getBehaviorDataMatrix(H, C);
        W := getWarningDataMatrix(H, C);
        G := getGraphDataMatrix(H, C);
        Return D F B W G;
    else
        Return D;
    end if
end while
```

**图 4-5 多维度数据视图静态数据加载伪代码**

#### 4.1.2.2 目标信息图谱实现

目标信息图谱的构建过程需要多次的迭代，需要增量式的数据叠加和历史数据梳理机制，即对各个时段的图谱片段进行整合。目标信息图谱中的各类感知数据索引，随着各类感知数据的不断累积，利用上述数据关联算法增量式的更新，对于新关联上的感知数据索引，需要在目标信息图谱中进行添加，而对于前期由于信息不完全导致的错误关联，则需要在目标信息图谱中进行删除，需要删除感知数据索引及其关联关系。当各类感知数据累积到一定阈值后，就会触发离线计

算中的特征提取和行为分析算法，从而不断的更新目标信息图谱中的特征和行为信息。目标信息图谱设计与演化模式如图 4-6 所示。

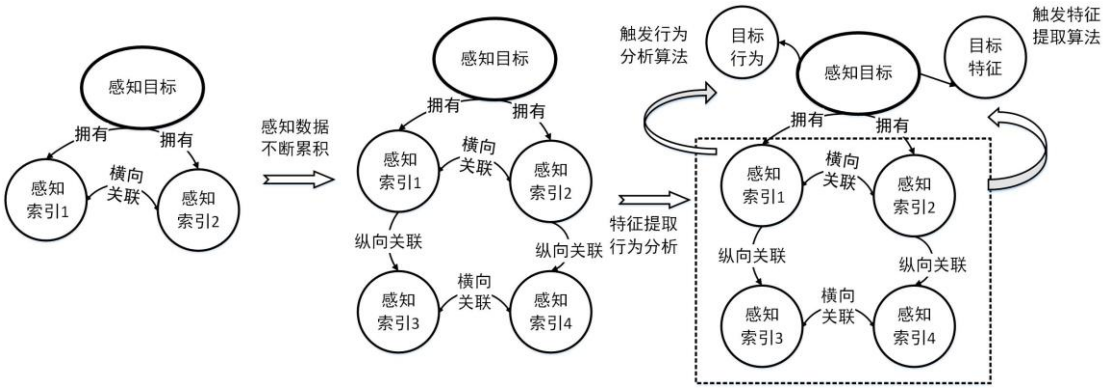


**图 4-6 目标信息图谱演化模式**

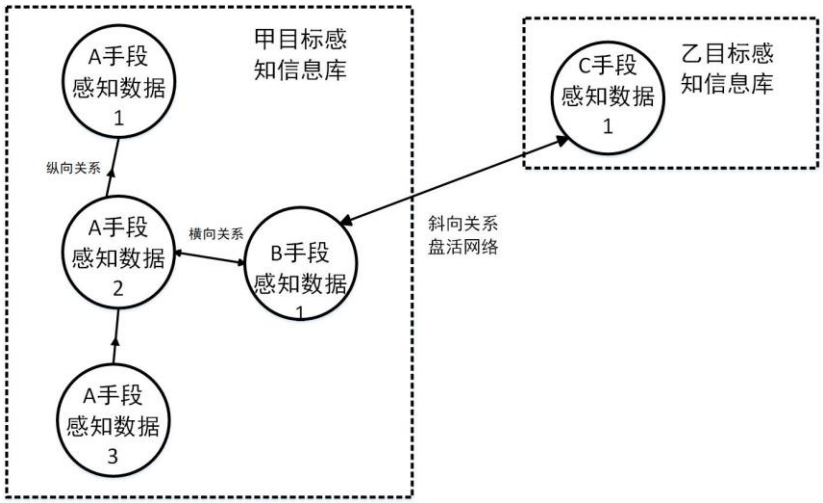


**图 4-7 目标信息图谱多关联关系及网络盘活**

目标信息图谱展示了同一个探测目标的多源异构数据之间的相关关联关系。比如电磁频谱、雷达、AIS 探测设备等设备同时探测到一个目标之后，设备的感知探测数据之间会产生横向、纵向、斜向的关系。同一目标不同设备在相同时间、相同地点的感知数据间产生横向相关关系。同一目标相同设备在不同时间相同地点探测到的感知数据间产生纵向相关关系。同一目标不同时间、不同地点由不同设备探测得到的感知数据产生斜向关系。目标信息图谱结合时空数据、环境信息、目标信息呈现了同一目标多维度设备数据通过横向、纵向、斜向串联，反复迭代，从点到树，从树到网的目标信息网结构。

目标信息图谱多关联关系及网络盘活如图 4-7 所示。发现目标信息图谱整体大图中不同凝集子群之间的斜向关系是盘活整个目标信息网结构的关键所在。例

如，对于甲目标，系统中存在包括 AIS、雷达等多种感知设备的情报信息。对于系统中发现的目标乙，我们只能仅仅从 C 手段感知的层面发现目标乙的感知情况。如果目标乙的 C 手段感知数据信息与目标甲的 B 手段感知数据信息证实具备关联性，是属于探测了同一个目标的信息，那么我们即发现了目标信息图谱中目标甲与目标乙之间的斜向关系，可以将目标甲与目标乙关联起来，证实了目标乙就是目标加。将两个凝集子群融合成为一个大群组，实现网络盘活。凝集子群之间的斜向关系有助于对目标进行快速而准确的认知，使有太多冗杂信息的多目标信息变得更加直观清晰[20,21,22]。

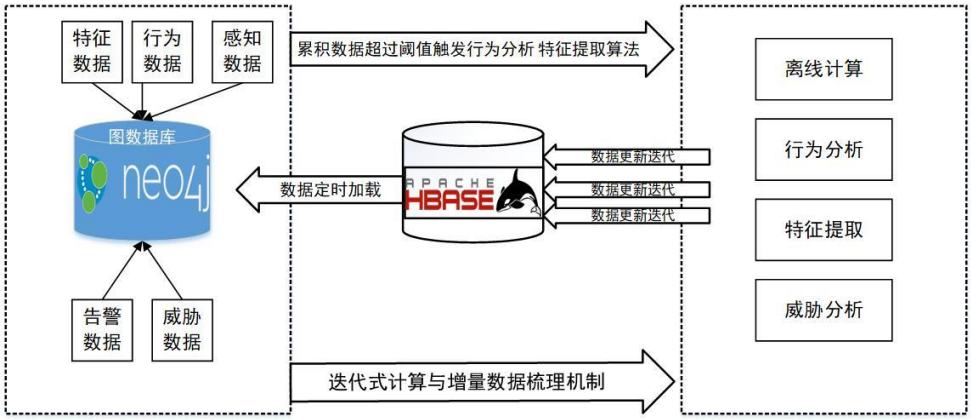


**图 4-8 目标信息图谱数据存储实现**

目标信息图谱数据存储实现如图 4-8 所示。我们通过离线计算、行为分析、特征提取、威胁分析后的目标多维度信息存入 HBase 列式数据库中，图数据库 Neo4j 存储了目标信息图谱，Neo4j 中存储的主要是目标节点与目标之间的关联关系数据。随着目标信息图谱的不断扩张，会产生越来越多的目标感知、特征、行为数据，以及具有明确威胁等级标签的目标数据。当目标感知、特征、行数据累积到一定阈值后，会触发离线计算中的数据挖掘算法，从而更新 HBase 数据库，进而更新整个目标信息图谱。例如目标在黑名单中，目标信息图谱一方面做出告警，另一方面将该目标的感知、特征、行为数据以及威胁等级持久化，从而供离线计算进一步的目标识别和威胁模型的训练。而对于无法明确进行目标识别和威胁分析时，则可以利用训练得到的威胁分析模型，以目标感知、特征、行为数据作为输入，对目标威胁等级进行预测。

Neo4j 目标节点中存储了 HBase 中详细多维度特征、行为、事件、威胁、感知数据在 HBase 中的行键索引，方便进行数据查询的预加载，查询具体多维度数据视图时，先从 Neo4j 中的节点中获取行键索引，再从 HBase 中获取具体的详细数据信息。从而实现离线与实时结合的目标信息图谱的存储。目标信息表如

表 4-1 所示，HBase 中的原始感知数据存储如表 4-2 所示。

**表 4-1 目标信息表**

| Table：目标信息表 | | |
|---|---|---|
| Row Key： | MB ID | |
| Family | MB | |
| Columns： | T1： | Value：不同的手段（AIS/Radar/光电），手段的 ID |
| | T2： | |
| | …… | |

**表 4-2 原始感知信息表**

| Table：AIS 航迹表 | | |
|---|---|---|
| Row Key： | MMSI 号+航迹起始点的时间戳 | |
| Family | AIS | |
| Columns： | T1： | Value：时间戳、经度、维度、速度、方向、关联的 Radar 航迹 ID 等 |
| | T2： | |
| | …… | |

| Table：Radar 航迹表 | | |
|---|---|---|
| Row Key： | 航迹起始点的 id | |
| Family | Radar | |
| Columns： | T1： | Value：时间戳、经度、维度、速度、方向、关联的 AIS 航迹 ID 等 |
| | T2： | |
| | …… | |

#### 4.1.2.3 设备显控联动与感知数据视图组件实现

原始感知数据视图组件是一个包含了三层嵌套组件组件集合。它配合完成了设备显控联动的功能目标的设备检测与监控。感知数据视图组件提供了关联到同一目标下的目标文本信息、可视化的视频、图像信息、地理标记信息的显示和控制，并且可以进行设备控制命令联动下发。原始感知数据视图组件从外部看，实际上是多维度数据视图组件的一个子视图组件。从其组件内部看，最外层为

TabPane 组件，中间层为 Collapse 组件与视频监控组件。Collapse 组件嵌套了包括 AIS 与雷达信息的 Table 组件。视频监控组件嵌套了一系列对接光电视频设备的 Button 组件，用于完成控制命令下发。

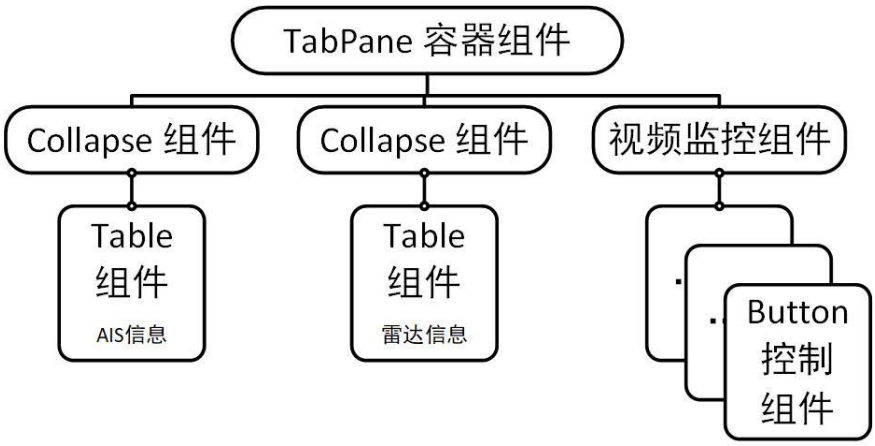


**图 4-9 感知数据视图组件结构**

Collapse 组件是一个可以折叠展开的内容区域，它包含了 AIS 与雷达表格信息，并且可以对信息表格进行分组和隐藏，用于保持面板的整洁。我们使用 Table 组件进行大量结构化的 AIS 与雷达帧结构信息展示，并且按照需要对结构化的感知信息进行排序、搜索、分页、自定义操作等复杂行为。当批量综合感知数据通过自适配的方法从多维度数据视图中流向感知视图组件后，通过感知视图的最底层 Table 组件的 dataSource 属性指定相应的 AIS 雷达数据信息，并通过 columns 属性指定表头与列配置信息，控制列的 className、对齐方式、默认排序等属性，通过 Table 组件，我们可以实时显示目标的径向距离、方位角、径向速度、经纬度等动态、静态感知信息。

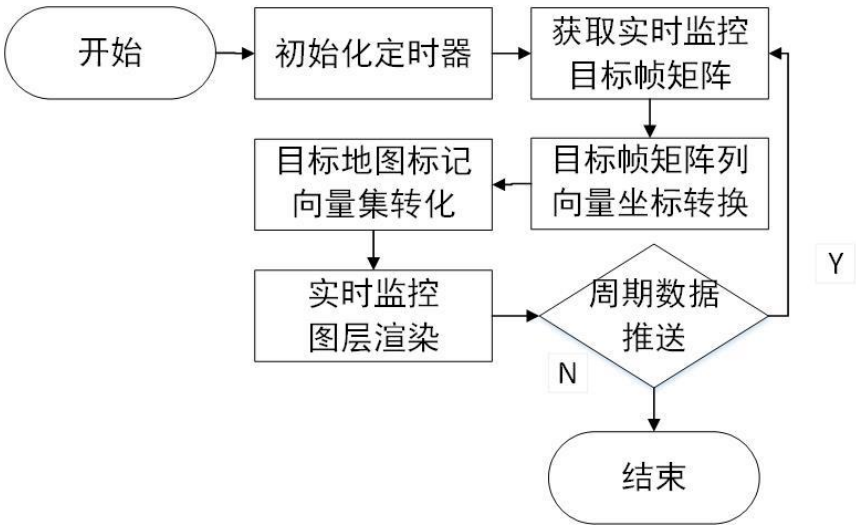


**图 4-10 设备显控渲染流程**

目标的原始感知位置信息通过 Supermap GIS 服务作为标记向量集合标记渲染在地图的实时目标显示图层上，系统通过数据定时器周期性的获取实时监控目标帧矩阵，这个矩阵中的每一行为当前目标的原始感知融合信息，包括了所有能关联在该目标下的设备的感知信息。开始实时监控模式后，首先初始化数据定时器，定义其数据请求频率，在每一个请求周期的服务端回调函数内进行该周期的标记向量图层渲染任务。回调函数开始时首先要清楚实时监控地图标记向量图层，用来保证渲染的实时性，获取到实时监控目标帧矩阵后，开始遍历矩阵的每一行，在遍历每一行的过程中，抽取每一个目标的经纬度信息，使用 SuperMap.Marker 类为每一个实时目标创建相应经纬度的 Marker 标记对象，遍历完成实时监控目标帧矩阵后，将获取到的 Marker 对象存储成为标记向量，并且将标记向量矩阵添加到实时监控图层中，完成实时监控图层的渲染。实时监控模式设备显控的渲染流程如图 4-10 所示。

**Algorithm 2** Realtime Dynamic Batch Position Frame Data Load and Render

**Input:**

The state of timer $S$. The matrix Batch Position data $D$.
The state of map type $T$. The static variable $\Phi$.

**Output:**

The matrix render marker $M$. The matrix render vector $V$.

Create a batch data heap $H$; Inital record index $E$;
**while** $(S == True)$ **do**
  $clearHeap(H)$;
  **while** $(getCurrentIndexDataRecord(H, E)! = null)$ **do**
    get current index data record $R$:=$getCurrentIndexDataRecord(H, E)$;
    get current lontitude latitude $L$:=$getCurrentLontitudeLatitude(H, R)$;
    **if** $(T == EPSG : 3857)$ **then**
      get current lontitude $L_{lon}$=$getCurrentLontitude(L)$;
      get current latitude $L_{lat}$=$getCurrentLatitude(L)$;
      transform current lontitude coordinate;
      $\widetilde{L_{lon}}$=$L_{lon}$*Φ /180 ;
      transform current latitude coordinate ;
      $\widetilde{L_{lat}}$=ln(tan(π$L_{lat}$/180+/pi/2)*/2)*Φ/π ;
      $L \leftarrow (\widetilde{L_{lon}}, \widetilde{L_{lat}})$;
    **end if**
    $M_E$:=$(getTargetRenderSupermapMarker(H, L, E))$;
    $M \leftarrow M_E$;
    $V_E$:=$(getTargetRenderSupermapFeatureVector(H, L, E))$;
    $V \leftarrow V_E$;
  **end while**
  Return $M, V$;
  $RenderMartrixMarkerVectorLayer(H, M, V)$;
**end while**

**图 4-11 动态位置帧数据加载与渲染算法**

如图 4-11 动态位置帧数据加载与渲染算法所示，进行实时监控设备显控渲染的过程中，需要根据当前地图的类型进行实时的坐标转换。$L_l$是 WGS84:World Geodetic System 1984 下的经度坐标，$L_a$是 WGS84 标准下的纬度坐标，$L_{tl}$是 Pseudo Mercator 投影坐标系下的经度坐标，$L_{ta}$是 Pseudo Mercator 投影坐标系下下的纬度坐标，R 为地球赤道周长的一半，转换公式如下所示。

$$L_{tl} = L_l \times R \div 180 \tag{4-1}$$

$$L_{ta} = \ln\left(\frac{tan\left(\frac{\pi L_{la}}{180}+\frac{\pi}{2}\right)}{2}\right) \times R/\pi \tag{4-2}$$

#### 4.1.2.4 基于主动深度置信网络的目标威胁估计模型设计

多维度多级数据库中存储了目标所有的多源异构感知数据与行为、特征、告警的数据集，由于目标的感知、轨迹行为、特征、告警数据集中含有大量的未标

注的数据，其中的存储的实际发现告警的具备威胁行为的目标占少数，而存在一些目标虽然没有产生设备告警，但是仍然具有威胁，实际上它的轨迹行为和一些特征具备一定的危险性，只是我们通过一些的机械的显式规则发现不了它。针对这种训练集中有大量的未标注数据，标注它们要耗费大量的时间和金钱的时候，我们需要使用半监督的深度学习方法进行威胁估计模型训练，并且通过主动学习的方法来确定需要标注的最难区分的数据，然后使用已经选择的标注好具备威胁的目标多维度数据与所有未标注的数据来训练深层架构，这个深层架构在主动学习的过程中不断的发现出来最难区分是否具备威胁的目标，并且进行威胁分析的标注，这样不断的进行迭代训练，逐步提高主动深度置信网络威胁分析模型的威胁估计能力。

将所有目标的感知、特征、告警、威胁数据视图按照时间采样后，取出一个时间帧的数据，每一个目标用一个向量表示，此时目标帧数据矩阵为：

$$\mathrm{X} = [x^1, x^2, \dots, x^{R+T}] = \begin{bmatrix} x_1^1 & \cdots & x_1^{R+T} \\ \vdots & \ddots & \vdots \\ x_D^1 & \cdots & x_D^{R+T} \end{bmatrix} \tag{4-3}$$

其中 R 是训练数据的数量，T 是测试数据的数量，D 是每个目标的特征维度个数，其中 L 个标注数据是从 R 个目标帧训练数据集中随机选择的或者用主动学习的方法主动选择，令 S 为从训练数据集中需要人工标注的目标索引集合。标注目标数据表示如下所示。

$$X^L = X^R(S),\ S = [s_1, \dots, s_L]\ 1 \le s_i \le R \tag{4-4}$$

令 C 为态势威胁估计的威胁程度类别，Y 是与 L 个标注数据所对应的标签数据集，其可以表示为如下所示。

$$Y^L = [y^1, y^2, \dots, y^L] = \begin{bmatrix} y_1^1 & \cdots & y_1^L \\ \vdots & \ddots & \vdots \\ y_C^1 & \cdots & y_C^L \end{bmatrix} \tag{4-5}$$

Y 的每一列是一个空间向量，坐标为 j 的值代表了第 j 类威胁程度。

$$\mathrm{y_j} = \begin{cases} 1 \text{ 如果 x 属于第 j 个类别威胁程度} \\ -1 \text{ 如果 x 不属于第 j 个类别威胁程度} \end{cases} \tag{4-6}$$

态势估计模型训练完毕后将一个新的目标时间帧向量数据输入 X 至 Y 的映射函数时，主动深度置信网络态势估计模型将会给出此目标的威胁程度预测。

主动深度置信网络的网络拓扑结构如图所示。它是一个全连接定向的多层神经网络，包括了一个输入层 $h^0$，N 个隐藏层 $h^1$ 至 $h^N$，顶部为输出层 f。其中我们需要训练的态势估计模型参数为$\mathrm{W} = \{w^1, w^2, \dots, w^{N+1}\}$。主动深度置信网络的架

构由限制玻尔兹曼机（Restricted Boltzmann Machines,RBM）构建起来。RBM 是一个两层的递归神经网络，它通过对称的权值把随机的二进制输入输出连接。

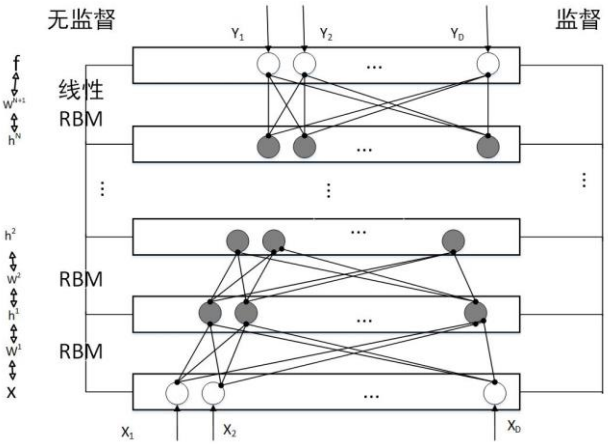


**图 4-12 深度置信网络结构**

在主动深度置信网络，能量状态公式如下所示。

$$\mathrm{E}(h^{k-1},h^{k};\theta)=-\sum_{s=1}^{D_{k-1}}\sum_{t=1}^{D_k}w_{st}^{k}h_s^{k-1}h_t^{k}-\sum_{s=1}^{D_{k-1}}b_s h_s^{k-1}-\sum_{t=1}^{D_k}c_t h_t \tag{4-7}$$

其中$\theta=$（w, b, c)是态势估计模型的参数，$w_{st}^{k}$是隐藏层 $h^{k-1}$ 的单元 s 和隐藏层 $h^{k}$ 的单元 t 之间的对称连接参数，k 区间为[1,N-1]。$b_s$ 是隐藏层 $h^{k-1}$ 中第 s 个偏置,$c_t$ 是隐藏层 $h^{k}$ 中第 t 个偏置。$D_k$ 为第 k 层结点数。

$h^{k-1}$ 发生的概率如下所示。

$$\mathrm{P}\left(h_t^{k-1};\theta\right)=\frac{1}{Z(\theta)}\sum_{h^k}\exp(-\mathrm{E}(h^{k-1},h^{k};\theta)) \tag{4-8}$$

归一化常数如下所示。

$$Z(\theta)=\sum_{h^{k-1}}\sum_{h^k}\exp(-\mathrm{E}(h^{k-1},h^{k};\theta)) \tag{4-9}$$

$h^{k-1}$ 与 $h^{k}$ 的条件概率如下所示。

$$\mathrm{p}(h^{k}|h^{k-1})=\prod_t p(h_t^{k}|h^{k-1}) \tag{4-10}$$

$$\mathrm{p}(h^{k-1}|h^{k})=\prod_s p(h_s^{k-1}|h^{k}) \tag{4-11}$$

第 t 单元为 1 的概率可用逻辑函数如下表示：

$$\mathrm{p}\left(h_t^k = 1|h^{k-1}\right) = sigm(c_t + \sum_s w_{st}^k h_s^{k-1}) \tag{4-12}$$

第 t 单元为 1 的概率可用如下逻辑函数表示：

$$\mathrm{p}(h_s^{k-1} = 1|h^k) = sigm(b_s + \sum_t w_{st}^k h_s^k) \tag{4-13}$$

逻辑函数如下所示。

$$\mathrm{sigm}(\eta) = 1/(1 + \exp(-\eta) \tag{4-14}$$

对隐藏层产生的概率对数求导数可以得到数据分布的期望与运行 Gibbs 采样 M 次之差。如下所示。

$$\frac{\partial logp(h^{k-1})}{\partial w_{st}^k} = \langle h_s^{k-1} h_t^k \rangle_{P_0} - \langle h_s^{k-1} h_t^k \rangle_{P_M} \tag{4-15}$$

参数 $w^k$ 可以通过如下公式调整。

$$w_{st}^k = \vartheta w_{st}^k + \eta \frac{\partial logp(h^{k-1})}{\partial w_{st}^k} \tag{4-16}$$

计算 $w^k$ 得到后，可以计算数据 x 从 $h^0$ 输入后的隐藏层公式，如下所示。

$$h_t^k(x) = sigm(c_t^k + \sum_{s=1}^{D_{k-1}} w_{st}^k h_s^{k-1}(x) \quad t = 1,2,\dots,D_k; k = 1,2,\dots,N-1 \tag{4-17}$$

使用正态分布对 $w^k$ 进行初始化后再运行梯度下降优化方法可能会带来危机。故主动深度置信网络使用线性 RBM 初始化输出层参数 $w^{N+1}$,线性 RBM 从服从高斯分布的随机变量中采样获得。

F 的第 j 个单元的值是一个线性函数，如下所示。此时该数值没有经过迭代。

$$f_{j,0} = c_j^{N+1} + \sum_i w_{ij}^{N+1} h_{i,0}^N \tag{4-18}$$

F 的状态值是 f 的值加上一个服从正态分布的随机变量。

$$s_{j,0} = f_{j,0} + r \tag{4-19}$$

其中 hN 的第 i 单元的新的值是包含了 f 状态值$s_{j,0}$和$w_{ij}^{N+1}$的逻辑函数，如下所示。

$$h_{i,1}^{N} = \mathrm{sigm}(b_i^N + \sum w_{ij}^{N+1} s_{j,0}) \tag{4-20}$$

F 第 i 个单元的新值如下所示。

$$f_{j,1} = c_j^{N+1} + \sum_i w_{ij}^{N+1} h_{i,1}^{N} \tag{4-21}$$

类似 RBM，线性 RBM 的求导公式如下所示。

$$\frac{\partial log p(h^N)}{\partial w_{ij}^{N+1}} = h_{i,0}^{N} f_{j,0} - h_{i,1}^{N} f_{j,1} \tag{4-22}$$

使用线性 RBM 训练得到 WN+1 初始化值，此时输出层 f 如下所示。

$$f_j(x) = c_j^{N+1} + \sum_{i=1}^{D_N} w_{ij}^{N+1} h_{i,1}^{N}(x) \quad j = 1,2,\ldots,C \tag{4-23}$$

输出层的损失函数如下所示。

$$\mathrm{E(z)} = \exp(-\mathrm{z}) \tag{4-24}$$

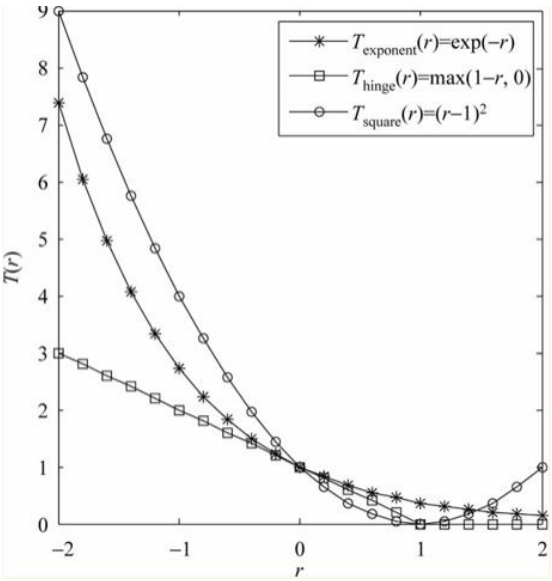


**图 4-13 损失函数**

如图所示，当 r 增大时，Tsquare 随着 r 增大而增加，故均方误差函数损失函数对正确分类的定义区间非常小，过于正确的分类将会得到分类错误率增加的惩罚。

指数损失函数的最优化问题总结为如下公式。监督学习阶段使用随机梯度下降算法优化整个深层架构的所有参数，无监督学习阶段的随机参数被确定实概率代替。

$$\arg_f min \sum_{i=1}^{L} l(f(x^i), y^i) \tag{4-25}$$

$$\mathrm{l(f(x), y)} = \sum_{\mathrm{j=1}}^{\mathrm{C}} \mathrm{E(f_j(x) y_j,)} \tag{4-26}$$

目标的帧数据矩阵 X 中包括了未标注的目标数据池 $X^R$ 与初始威胁目标标注数据集 $X^L$。深层架构 $h^N$ 将确定 $X^R$ 中的哪个未标注目标进行手工标注，在这之后 $h^N$ 的参数将会被增加了手工标注进行主动学习后的新训练集进行优化。目标经过映射后在未标注数据池中与类分界之间的距离如下所示。

$$d^i = \left| h_1^N(x^i) - h_2^N(x^i) \right| / \sqrt{2} \tag{4-27}$$

从而选择出的需要主动标注器威胁程度的目标如下。

$$\mathrm{s = \{j: } d^j = \mathrm{min(d)\}} \tag{4-28}$$

实际上目标的威胁程度由深层架构的输出的位置决定。通过交替使用无监督学习和监督学习训练的方式，更好的调整了神经网络深层架构，提高威胁估计模型的威胁判别分析能力，经过主动学习后的深度置信网络成为提供威胁分析数据视图的模型。其训练算法如下图所示。

**Algorithm 1** 主动深度置信网络态势威胁估计模型训练算法

**Input:**

数据集 X,$(X^L, Y^L)$ 层数 N, 无监督学习迭代次数 $Q$

训练数据个数 R，测试数据的个数 $T$

服从正态分布的随机初始化参数空间 $W$

主动学习迭代次数 $I$

主动学习每次迭代选择的数据向量数 $G$

**Output:**

包含训练后参数空间 $W$ 的深层架构

1: **for** $i = 1; i <= I$ **do**
2: 贪心无监督方法一层层构建网络;
3: **for** $q = 1; q <= Q$ **do**
4: **for** $k = 1; k <= R + T$ **do**
5: 计算非线性正向和反向状态;
6: $p(h_t^k = 1|h^{k-1}) = sigm(c_t^k + \sum_s w_{st}^k h_s^{k-1})$
7: $p(h_t^{k-1} = 1|h^k) = sigm(b_s^{k-1} + \sum_t w_{st}^k h_t^k)$
8: 更新参数和偏置
9: $\frac{\partial log p(h^{k-1})}{\partial w_{st}^k} =< h_s^{k-1}, h_t^k >_{p_0} - < h_s^{k-1}, h_t^k >_{p_M}$
10: **end for**
11: **end for**
12: 使用线性 RBM 构建输出层
13: 基于梯度下降的监督学习
14: 在标注集 $X_L$ 上最小化损失函数，更新参数空间 W
15: 选择 G 个最难区分的数据输入向量：$s = j : d^j = min(d)$
16: 添加选择的 G 个文档到标注数据集 $X_L$
17: **end for**
18: 迭代进行 RBM 构建与无监督方法网络构建

**图 4-14 网络模型训练算法**

#### 4.1.2.5 实时告警与威胁数据视图组件实现

威胁数据视图组件是一个三层嵌套树形组件。它展示了目标的威胁分析视图的数据，分为目标的威胁等级、目标威胁判据及告警跃迁的过程。由于机器无法判定是敌是我，故只能给出置信度与判据，最终还是交由人来判断。威胁数据视图组件内部下一层为一个 Card 组件，用来作为组件容器，最低层为 Rate 组件与 Timeline 组件，分别用于展示目标威胁等级与威胁判据及告警跃迁的过程。威胁数据视图组件结构如图 4-15 所示。

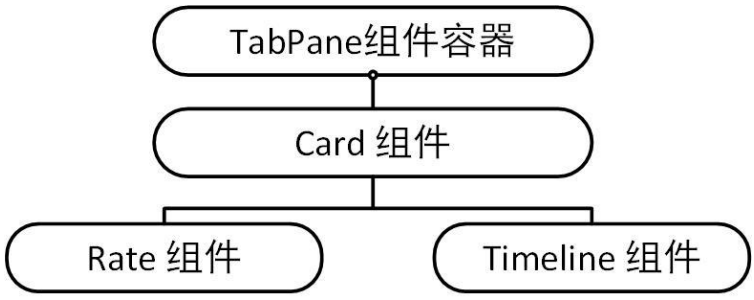

**图 4-15 威胁数据视图组件结构**

Rate 组件作为目标的威胁等级的评分组件，可以对目标的威胁评价进行展示。Timeline 组件用于垂直展示告警跃迁时间流信息。系统的告警跃迁过程实际上是目标威胁判据的计算过程的详细展现，目标的告警等级并不是静态的，并不是根据发生的危险行为、告警事件得到的静态结果。而是发展的、动态的，所有目标在数据刚入库时告警等级均为正常，也就是没有威胁，在长期对其进行监控的过程，中由于目标发生了一些危险行为与告警事件而使其威胁不断提升，比如，目标在某时段作为非协作目标进出了监控预警区，使其威胁等级上升为轻度警告，又在某时段发生了 ais 恶意篡改事件，从而进一步提升了威胁等级，产生了告警跃迁。

探测目标到底是暴风雨中停靠的一只小船，还是中北美的加勒比海盗，依赖于我们对于目标的威胁识别分析，我们从防护区、聚集区等区域角度结合历史挖掘结果，对于目标的历史异常行为，比如 AIS 静默、AIS 伪装篡改行为进行记录，并且形成目标的黑名单，在可视化效果方面，体现为使用 Timeline 组件进行时间轴上的串联，即将目标的历史危险行为与异常检测发现算法的告警跃迁流程串联起来，这本身既是目标告警跃迁的过程，又形成了目标的威胁判据。

实时监控模式下，一旦目标出现实时告警事件，目标渲染标记将会根据目标告警等级实时更新渲染颜色，并且将根据告警程度不不同进行不同频率的闪烁，以智能化、形象化的表征设备警报故障与目标告警信息。图 4-16 展现了实时告警闪烁流程。

实时监控模式的实时告警功能通过多定时器的方式进行实现，首先进行告警定时器和数据定时器的初始化，告警定时器负责告警目标标记的实时颜色变化渲染任务与告警目标以不同频率进行闪烁的功能。数据定时器负责周期性的感知数据帧矩阵的数据请求任务。数据定时器请求了感知数据帧矩阵后，将当前实时感知目标中的具备不同告警等级的目标抽取出来，然后所有不同告警等级的目标感知帧矩阵数据传输给相应的告警定时器。告警定时器将以不同于数据定时器的频率进行工作，也就是说告警定时器与数据定时器是不同步的，告警定时器工作频率是数据定时器的整数倍，并且告警目标威胁等级越高，告警定时器工作频率越快，告警定时器会根据工作频率的不同给与目标标记不同程度的颜色渲染。告警目标闪烁的功能通过告警定时器延时半周期渲染的方式实现，也就是说，在告警定时器的一个工作周期内，一半时间用来渲染数据，一半时间隐藏数据标记，这样就实现了闪烁效果，由于工作频率越高，告警目标威胁等级越高，这样也就

实现了威胁等级越高，告警目标地图标记闪烁越快。实时告警闪烁算法如图 4-17 所示。

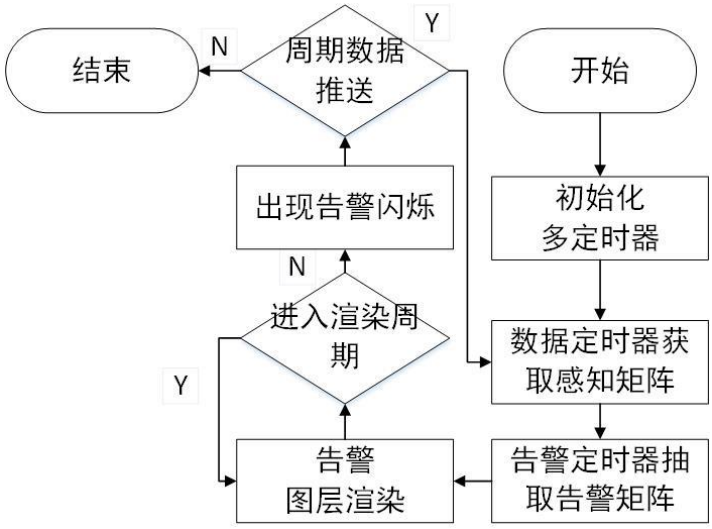


图 4-16 实时告警闪烁流程

**Algorithm 3** Realtime Warning Data Render Flicker

**Input:**
The state of warning timer $S_w$. The state of data timer $S_d$.
The waring timer array $T$. The waring level array $A$.
**Output:**
The matrix warning data $W$.

Create a data Heap $H$;
**while** $(S_w == True || S_d == True)$ **do**
  $clearHeap(H)$;
  **if** $(S_d == True))$ **then**
    get current data matrix $M$:=$getTargetData(H)$;
  **else**
    get current warning timer$T_c$:=$getCurrentWarningTimer(H, T)$;
    get current waring level $A_C$:=$getTargetData(H, A)$;
    **while** $(getNextWarningData(T_c, A_C, M)! = null)$ **do**
      get current waring level data $W_c$:=$getCurrentWaringLevelData(T_c, A_C, M)$;
      $W \leftarrow W_c$;
    **end while**
    Return $W$;
    $addWarningDatatoRenderLayer(W, H)$;
    $setTimeoutRenderWarningData(W, H, T, A)$;
  **end if**
  **if** $getTimeoutStateFlickerRender(T, A) == True$ **then**
    $FlickerRenderWarningData(W, H)$;
  **end if**
**end while**

图 4-17 实时告警数据渲染闪烁算法

#### 4.1.2.6 基于概率规划推理的目标行为分析

目标的历史行为将会存储在多级数据库中，它们的历史行为将会直接参与目标实时行为的分析，从而进行决策。目标实时行为分析基于期望效用最大化原则，同时考虑行为目标的目标合意性与目标实现的可能性。在计算行为的期望效益值时候要考虑目标行为后果的效用值来表示目标合意性和行为后果出现的概率。

我们设$P_{STATE}$为行为状态的概率，作为行为前提的不确定性。设$P_{EXECUTION}(A|precondition(A))$为行为执行的不确定。设$P_{EFFECT}(e|A)$表示当行为 A 成功执行时候，该行为结果出现的概率。

首先介绍目标行为状态概率的计算。假设 E 为目标行为证据，若在目标行为证据 E 下观察到的状态 x,则其概率为 P（x|E）=1。我们监控到的目标行为会改变与其进行关联的状态出现的概率。如果观察到目标正在执行的或者目标已经的执行的执行行为 B，那么目标行为 B 出现的每个前提的概率为 1。如果正在目标执行行为为 A，则目标行为 A 的每个结果 e 出现的概率为其执行概率$P_{EXECUTION}(A|precondition(A))$与其结果概率的$P_{EFFECT}(\mathrm{e}|A)$的乘积。如果目标执行行为为 A，则行为 A 的每个结果 e 出现的概率为$P_{EFFECT}(\mathrm{e}|A)$。否则目标行为状态 x 出现的概率等于 x 的先验概率值 p(x)。

目标行为概率值的计算。给定目标实时行为证据 E，如果观察到已经目标执行了行为 A，则 P（x|E）=1；如果监控到目标正在执行行为 A，则 P（A|E）等于目标行为执行概率。否则目标行为 A 发生的概率都等于目标行为 A 的执行概率与它的每个目标行为前提出现的概率的乘积。公式如下。

$$\mathrm{P}(\mathrm{A}|\mathrm{E}) = P_{EXECUTION}\left(A\middle|\prod_{e\in precondition(A)} precondition(A) \times P(e|A)\right) \quad (4\text{-}29)$$

目标行为概率的变化对于目标行为结果概率的影响如下。

$$\mathrm{P}\ \mathrm{P}(\mathrm{e}|\mathrm{E}) = \mathrm{P}(\mathrm{A}|\mathrm{E}) \times P_{EFFECT}(e|A) \quad (4\text{-}30)$$

目标行为后果概率与目标期望效用值的计算，目标行为概率的变化会影响目标行为结果概率，进而影响目标行为的期望效益值。设 O 为目标行为 A 的目标行为后果集合且其中一个目标行为后果为$o_i$。则：

$$P_{action}(o_i|\mathrm{E}) = \mathrm{P}(\mathrm{A}|\mathrm{E}) \times P_{EFFECT}(o_i|A) \quad (4\text{-}31)$$

行为 A 的期望效用值由 A 的每个后果出现的目标行为概率以及目标行为效用值计算得到：

$$\mathrm{EU}(\mathrm{A}|\mathrm{E}) = \sum_{o_i \in o_A} (P_{action}(o_i|\mathrm{E}) \times Utility(o_i)) \quad (4\text{-}32)$$

目标规划后果概率与目标期望效用值。目标行为概率的变化对于目标规划概率产生影响，进而影响整个目标行为分析规划的期望效益值。令 Op 为目标行为分析规划的后果集合，其中一个目标行为后果为$o_j$。设{A1,…,Ak}是导致目标行为规划中导致后果为$o_j$的偏序目标行为集合。Ak 导致了目标行为结果$o_j$。目标行为规划中导致$o_j$的所有目标行为与证据 E，再加上目标行为结果$o_j$的结果概率乘积确定了目标行为结果$o_j$出现概率。公式如下：

$$P_{plan}(o_j|\mathrm{E}) = (\prod_{i=1,\ldots,k} \mathrm{P}(A_i|\mathrm{E})) \times P_{EFFECT}(o_j|A_k) \quad (4\text{-}33)$$

目标行为规划期望效用值由目标行为规划中每个行为规划后果出现的概率以及效用值得到：

$$\mathrm{EU}(\mathrm{P}|\mathrm{E}) = \sum_{o_j \in o_p} (P_{plan}(o_j|\mathrm{E}) \times Utility(o_j)) \quad (4\text{-}34)$$

#### 4.1.2.7 情景回放与行为事件数据视图组件实现

行为事件数据视图组件是一个四层嵌套树形组件。行为事件数据视图展示了当前实时监控模式下的目标历史上的历史经过区域的行为与发生过的告警事件。组件本质上由两个 Collapse 组件构成，每个 Collapse 组件都包含了一个 Panel 面板组件用于容纳行为事件数据，Panel 组件下容纳了 Table 组件与 Button 组件，用于使用结构化表格展示行为事件数据以及进行选中后一键的行为事件情景回放。行为事件数据视图组件结构如图 4-18 所示。

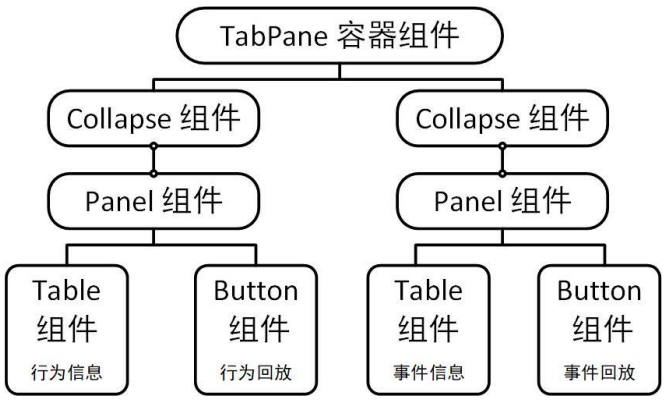


**图 4-18 行为事件数据视图组件结构**

在实时态势监控模式中，目标行为数据视图实时的呈现了目标历史发生的进出防护区行为、告警事件等历史情景，此时可以选中需要进行回放的告警事件或区域行为，通过历史回放服务暴露出的 API（Application Programming Interface 应用编程接口）进行无缝服务切换，一键式的从高维度实时态势监控模式的切换至历史回放服务。行为事件情景回放是基于实时与历史结合的思想，是实时模式与历史模式的关联，用历史事件、行为情景辅助实时监控的思想，实现高维度、一键式基于实时历史无缝服务切换的实时态势监控情景回放。实时历史服务之间通过网络调用进行通信，加强了服务之间的隔离性，避免紧耦合。实时监控与历史回放服务切换如图 4-19 所示。

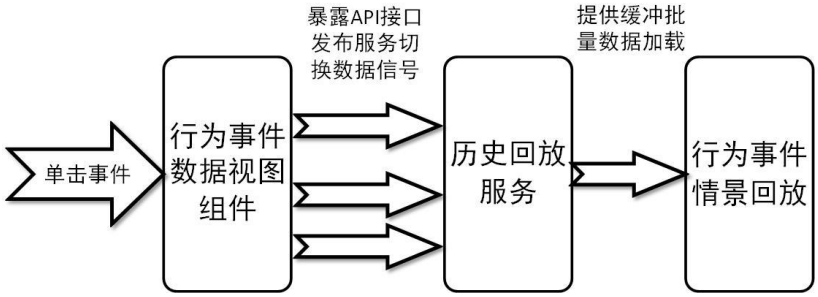


**图 4-19 实时监控与历史回放服务切换**

在查看行为事件数据视图的过程中，组件订阅了选定目标标记点的行为事件数据，数据通过自适配的方式直接推送给 Table 组件的 dataSouce 属性，通过 Table 组件的 column 属性设置目标结构化行为、事件信息列名。事件信息包括了告警事件名称、告警事件发生时间，告警目标 ID。行为信息列包括进入区域时间、离开区域时间、区域代号 ID 或者区域经纬度、区域停留时间、行为目标 ID。Table 组件通过 onChange 回调函数进行表格组件的实时渲染刷新。

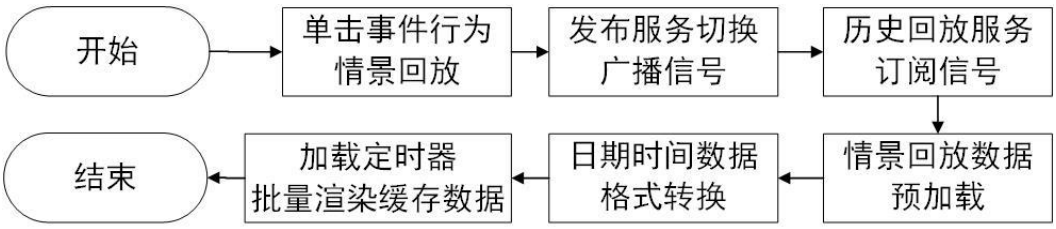


**图 4-20 情景回放服务切换流程图**

当对 Table 组件中所展示的每一行具体事件或者目标行为记录进行选择、取消等复杂行为操作时，使用 Table 组件中的 onSelect 函数作为回调暂存下当前选中记录行为事件的信息。当对开始回放 Button 组件产生单击事件后，Button 组

件将执行 onClick 函数，将暂存的目标行为、事件情景记录关键信息封装入组件广播信号中，并且给信号广播器发布服务切换广播信号。由于历史回放服务订阅了服务切换信号主题，并且历史回放服务暴露出了进行历史回放所需要的 API，使用 API 并且传递指定的具体关键时空参数既可以启动历史回放服务，使其批量提供我们需要的缓冲数据和回放可视化方案。历史回放服务暴露出的 API 中所需的关键参数包括进行历史回放的 Mode、历史回放的起止 datetime（YYYY:MM:DD XX:XX:XX 格式），可选参数包括历史回放的区域、历史回放的事件 ID、历史回放的目标 ID、目标列表等。情景回放的服务切换流程如图 4-20 所示。

当历史回放服务接受到其订阅的服务切换主题信号后，首先通过 API 进行行为、事件回放数据的预加载。预加载分解为两步，第一步将 HBase 中存储的历史回放起止时间戳内的轨迹感知数据行键索引进行预先准备，使用空间换时间的思想，具体进行历史批量数据缓冲时将根据批量的轨迹行键索引直接映射至数据库具体对象中进行快速批量数据查询。回放数据的预加载第二步为预加载告警事件、行为关键时间帧节点基础信息，包括了告警事件、行为的关键帧的时间位置与其包含的事件行为类型编码、标识、关联目标 ID 等。历史回放服务经过情景回放数据预加载后首先将预加载的告警事件、行为关键时间帧节点基础信息提供给情景回放控制交互组件，控制交互组件使用 moment.js 进行日期时间的管理并且将日期格式的关键时间帧数据转化为表征进度的整数格式位置帧 value，通过位置帧 value 使进度 Slider 组件在相关位置使用红色标记出关键时间帧节点，并且在红色标记下面注明告警事件行为的简略文字信息，最终通过鼠标事件单击开始回放按钮加载定时器并且批量缓冲渲染缓存数据，完成情景回放。上述情景回放服务切换的伪代码如图 4-21 所示。

**Algorithm 4** Scenario Playback Service Switching

**Input:**

The state of playback mode $S$. The start date and time of scenario data $O$. The end date and time of scenario data $E$. The Identification scenario $D$.

**Output:**

The matrix playback batch data $M$.

1: Create a playback queue $Q$;
2: $publishServiceSwitchSignal(S, O, E, D)$;
3: Subscribe playback signal messagge $G$ :=$getSubscribePlackbackMessage()$;
4: $G \leftarrow S, O, E, D$;
5: $M \leftarrow preLoadEventInfo(S, O, E, D)$;
6: $waitingCallbackMessage()$;
7: transform date time format;
8: get transform start value $O_v$:=$transformTimeFormat(O)$;
9: get transform end value $E_v$:=$transformTimeFormat(E)$;
10: $setPlackbackState(O_v, E_v, M)$;
11: $StartPlayback(M)$;
12: Return $M$;

图 4-21 情景回放服务切换伪代码

### 4.1.3 历史数据回放模式设计与组件化实现

#### 4.1.3.1 缓冲式批量数据加载

为了实现历史数据回放，必须解决数据加载和速度控制问题。历史回放模式使用了一种缓冲式批量数据加载算法解决缓冲状态下的批量数据加载和速度控制问题，缓冲式批量数据加载算法如图 4-22 所示。

该算法创建缓冲队列 Q 以存储每次对应的回放目标信息数据。队列的索引值是每个帧数据时间相对于批量数据加载 O 的起始帧时间的偏移（转换为秒）。当用户点击开始，加速，减速或拖动进度条时，触发点击事件并打开计时器，然后将当前播放的状态设置为真。接下来，记录当前回放时间，并计算当前时间和回放开始时间之间的偏移，根据该偏移访问回放数据（*getBufferQueueElement(Q，C)*）。如果存在重放数据，则系统将在当前时间重放目标信息数据（*renderFrameData(getBufferQueueElement(Q，C))*）。如果数据不存在，系统将开始批量缓冲下一个缓冲时间窗口内的所有数据（*bufferBatchData(O，E，D，r)*）。最后，在缓冲完成后，它将回放当前时间的数据并加上相邻帧时间 D 的差值，并等待直到定时器的下一个循环时间重新确定当前回放 S 的状态是否继续。

历史回放速度由加速、减速按钮控制，当单击加速或者减速按钮后，系统会修改播放倍率，一个回放时间单位等于播放倍率与缓冲时间常量的乘积，通过控制回放时间单位的方法加大或减少批量加载数据的时间间隔，从而实现历史回放速度的控制。

**Algorithm 5** Buffer Batch Data Loading Algorithm

**Input:**
The state of current playback $S$.
The start frame time of batch data loading $O$.
The end frame time of batch data loading $E$.
Play speed factor $r$.
The difference of adjacent frame time $D$.

**Output:**
The matrix Batch data $M$.

```
Create a buffer queue Q;
while (S == True) do
    get current time C := getCurrentTime();
    if (getBufferQueueElement(Q, C) == null) then
        while (getBufferQueueElement(Q, E) == null) do
            Q ← bufferBatchData(O, E, D, r);
        end while
        renderFrameData(getBufferQueueElement(Q, C));
        M := getBatchDataMatrix(Q, O, E);
        Return M;
    else
        renderFrameData(getBufferQueueElement(Q, C));
    end if
    C := C + D;
end while
```

**图 4-22 缓冲式批量数据加载算法**

在实际的历史回放环境中，我们需要定时加载需要历史回放的帧数据，但是如果每个回放时间节点都向后台请求该时刻的目标回放帧数据会导致系统进行I/O 密集型的数据查询，增加系统负担，带来了大量冗余的网络流量，降低整个系统历史数据回放的流畅性，减少了回放效率。为了解决这一问题，我们采用定时缓冲批量数据的方法，在缓冲过程中，系统将一次性缓冲多帧数据，每一帧包括了所有监控目标在当前回放时刻的融合感知数据，这样降低了系统负担，减少冗余网络流量，可以增加历史数据回放的流畅性。

在历史回放过程中,缓冲批量数据的时间对于播放的效果非常重要，假定$T_i$为缓冲数据过程中缓冲第 $i$ 帧数据需要的时间，$T_b$为缓冲数据的总体时间，$w$ 为需要缓冲的帧的数量，那么缓冲一次批量数据的时间如公式所示。

$$T_b = \sum_{1}^{w} T_i \tag{4-35}$$

在实际的网络环境中，假设$T_{net}$为当前稳定的网络带宽条件下，传输 $w$ 帧数据需要的理论时间，$U_i$为传输相邻两帧数据之间的的帧延时，$\overline{T_b}$为缓冲一次批量数据的平均时间，那么缓冲过程的平均时间公式（4-4）所示。

$$\overline{T_b} = T_{net} + \sum_{1}^{w} U_i \tag{4-36}$$

在实际网络环境中，网络延迟及其他网络诱因呈现随机性，为了近似的计算平均缓冲时间，可以加入一个因子平滑帧延迟及延迟波动，可以使用历史值占较大比重，当前值占较小比重的方法估计缓冲时间，假定$U_f$为帧延时在 f 时刻的历史值，$U_l$为帧延时在 $l$ 时刻的当前值，且 $l$ 大于 $u$，$|U_f - U_{f-1}|$为延时波动的历史值，$|U_l - U_{l-1}|$为延时波动的当前值，γ为我们定义的平滑因子，那么平均缓冲时间为公式所示。

$$\begin{cases} \overline{T_b} = T_{net} + \gamma U_f + (1-\gamma) U_l \\ +\gamma \left| U_f - U_{f-1} \right| + (1-\gamma) \left| U_l - U_{l-1} \right| \\ \left| \gamma - \frac{3}{4} \right| < \frac{1}{4} \end{cases} \tag{4-37}$$

缓冲时间对于播放的效果，若缓冲时间窗口减小，则降低播放流畅性，若其增大，会增加回放延时性。所以缓冲时间就是需要根据网络延迟及延迟波动进行调整。

对于播放速度控制问题，我们通过控制速度调节因子从而控制相邻两帧包含的融合感知数据记录的时间差，假定$T_u$为帧数据时间差，$speed$ 为播放速度因子，$c$ 为我们定义的时间常量，那么帧数据时间差如公式所示。

$$T_u = speed \times c \tag{4-38}$$

历史回放速度由加速、减速按钮控制，当单击加速或者减速按钮后，系统会修改播放倍率，速度因子越大，两帧所记录的融合感知数据之间的时间差越大，实际的回放速度也就越快，从而实现历史回放速度的控制。

#### 4.1.3.2 历史数据回放方式组件实现

控制区的历史回放组件实际上是一个双层嵌套组件，它分为了全景回放组件、区域回放组件、事件回放组件、目标回放组件。我们可以根据查询需要，选择不同的回放模式，系统支持全景、个体、事件、区域四种不同的回放模式。控制区历史回放组件结构如图 4-23 所示。


**图 4-23 控制区历史回放组件结构**

四种历史回放方式组件结构如图 4-24 所示。

针对全量目标信息、整体态势、大面积全量环境的回放要求，系统提供了全景回放方式，将选定时间段的所有目标的全量信息按照时间维度的流失逐帧全信息回放。控制区域全景回放组件由 Model 组件与 li 标签组成，li 标签绑定了 clickHandle 函数负责打开 Model 对话框组件，用于启动历史回放服务，历史回放已经开始的情况下，再次单击 li 标签，将会停止历史回放服务，此时用于渲染批量缓存数据的定时器将会被 clearInterval 状态转移函数进行定时器状态清空并且进行定时器删除，释放出 javascript 堆空间，然后将历史回放标记向量渲染图层进行清空，确保地图上无历史回放的数据标记向量点。Model 组件是一个事务处理对话框容器组件需要用户处理事务，又不希望跳转页面以致打断工作流程时，可以使用 Modal 在当前页面正中打开一个浮层，承载相应的操作。Model 组件提供进行历史回放之前的批量数据查询条件的选择，全景回放 Model 组件包括了 RangePicker 组件与两个 Button 组件。RangePicker 组件与 DataPicker 组件之间具备继承关系，通过它可以进行 YYYY:MM:DD XX:XX:XX 格式的起止日期格式时间段的选择，可以精确到秒，当用户需要输入一个日期时候，可以点击选择起止时间后，弹出日期浮层面板对于全景回放的起止日期进行选择。通过 RangePicker 组件的 onPanelChange 回调函数存储当前选择的起止日期。单击开始回放按钮后，会调用历史回放服务的 API 接口，发送当前回放的起止时间与回放模式编码，历史回放服务根据当前的全景回放需求进行数据预加载。预加载分解为两步，第一步将 HBase 中存储的历史回放起止时间戳内的轨迹感知数据行键索引进行预先准备，使用空间换时间的思想，具体进行历史批量数据缓冲时将根据批量的轨迹行键索引直接映射至数据库具体对象中进行快速批量数据查询。

回放数据的预加载第二步为预加载告警事件、行为关键时间帧节点事件信息，包括关键帧的时间位置与其包含的事件行为类型编码。历史回放服务经过数据预加载后在回放流程控制组件生命周期 componentDidMount 阶段预加载的关键时间帧节点基础信息。然后通过历史回放流程控制组件进行全景回放的控制。

针对确定的单目标与多目标详细时段内历史感知信息的回放要求，系统提供目标回放的方式，针对选定的特殊单目标或多目标进行时间维度具体时段内的历史数据回放。目标回放组件与全景回放组件类似，仅仅是 Model 组件中增添了一个 Transfer 组件。Transfer 穿梭框组件用占据更大的空间，可以对将要回放的目标列表进行展示，可以使用直观的方式在左右两栏中移动元素，完成单目标或者多目标的选择行为。目标回放组件被选中时，向服务端请求当前目标信息库中的所有目标 ID 号，使用 dataSource 参数将目标 ID 数据源渲染到左边的 source 数据栏中，当用户选择了需要进行回放的目标后将目标 ID 穿梭添加到右方 target 回放目标栏中后，使用 targetKeys 参数获取右边回放列表的回放 ID 目标集合，再结合 RangePicker 组件获取的起止时间数据，具体使用时可以通过搜索框的 onSearch 搜索回调函数搜索出需要回放的目标 ID，并将其穿梭到右栏中，然后使用历史回放服务获取预加载数据并进一步实现单或多目标历史数据回放。

针对空间区域划分回放、防护区域历史事件的回放要求，系统提供区域回放方式，针对特定防护区或者鼠标绘制任意矢量区域图形内部的特定时段历史数据进行回放。其 Model 组件中增加了一个 Table 组件用于显示当前划分的防护区域的名称、范围、威胁程度，可以根据已经规划完毕的防护区进行回放，另外增加了一个 Button 组件，使用单击事件后通过 Supermap.Rectangle 类结合鼠标事件绘制出任选的一个矩形区域，并保存其经纬度坐标点，将其坐标点作为参数通过 API 向历史回放服务请求查询空间区域维度的缓冲批量数据以及进行数据预加载，进一步实现区域数据历史回放。最终然后通过历史回放流程控制组件进行区域数据历史回放的控制。

针对告警事件、异常数据检测的回放要求，系统提供事件回放的方式，针对历史发生的设备报警、目标行为告警、伪装告警等事件进行具体时段的事件回放。事件回放组件中增加了 Table 组件用于展示关键关键事件，类似于实时监控模式的情景回放组件，Table 组件中使用分页模式加载了所有的关键告警行为与告警事件，通过 RangePicker 组件进行日期索引搜索至一段时间内的所有关键告警行为与告警事件，然后选中具体的事件进行历史回放。

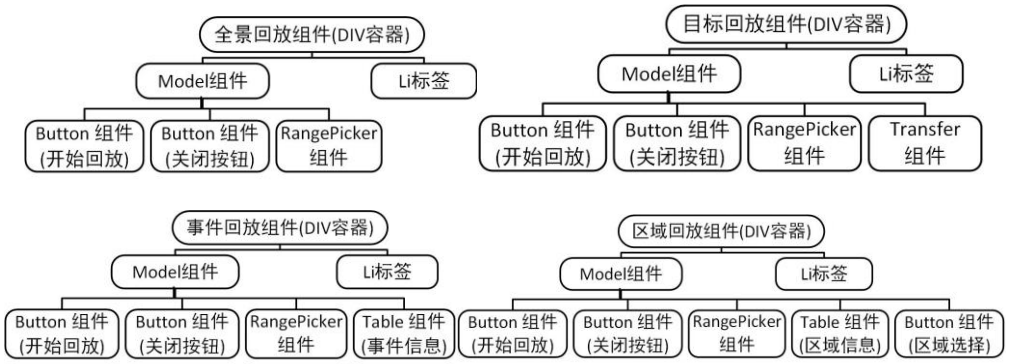


**图 4-24 四种历史回放方式组件结构**

#### 4.1.3.3 历史回放流程控制组件实现

历史回放流程控制组件是一个三层嵌套组件，它实现了历史回放的进度控制，经过了数据预加载后，历史回放正式进入回放进度流程，流程控制组件可以控制历史回放的开始、停止、暂停、加速、减速，并且有一个含有关键帧信息的进度条，可以进行进度条拖动，实现实际回放流程正向更新进度条与进度条反向控制回放流程，组件还可以显示回放的进度、当前回放时间、当前回放速率、缓冲状态等信息性的数据。组件实际上是一个 DIV 容器，它由 Card 组件作为回放信息容器，包含了三个 Button 组件，分别作为加速、减速、起止按钮，还包含了一个 Slider 组件作为进度条。历史回放流程控制组件结构如图 4-25 所示。

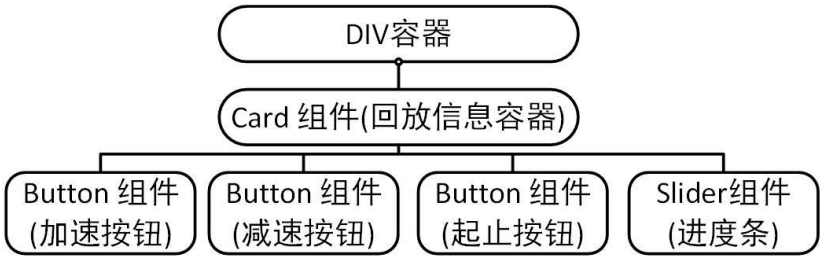


**图 4-25 历史回放流程控制组件结构**

Slider 组件的原型实际上是一个滑动输入器，可以展示进度的当前值与进度的范围数值自定义区间，并且可以进行自由选择滑动，它所表示的进度值既可以是连续的也可以是离散的。我们设计了一个能够展示时间维度回放进度的进度条 Slider 组件，通过其 step 参数设置步长为一秒，也就是说 Slider 组件每滑动一个单位，相当于时间走了一秒，相应 Slider 的 value 值也增加 1。以时间维度作为回放进度的组件存在一个最大回放区间，这个回放区间即回放数据进行预加载后

返回的总体回放的开始与结束时间，我们需要对开始和结束日期时间使用 moment.js 进行求差值处理，计算出开始与结束时间之间的时间差并且对这个时间差进行日期时间转换，使其从 YYYY/MM/DD HH:mm:ss 格式的时间转化为以秒为单位的时间，通过设置 max 参数将转换单位后的时间差值作为 Slider 组件的 value 最大上界，其 min 参数用于设置为零作为回放进度下界。

Slider 组件使用 tipFormatter 组件内部函数格式化 Tooltip 进度条提示的内容，当拖动进度条时，进度条滑块上方会显示出提示浮层，可以展示当前拖动进度回放的具体时间，进度条滑块拖动后会触发 onChange 事件，并且指示进度条 Slider 组件当前进度的 value 值传入，在进度条拖动触发 onChange 事件后，使用 tipFormatter 函数借助 moment.js 将以秒为单位的代表了当前进度值的 value 转换为实际的 YYYY/MM/DD HH:mm:ss 格式的时间，这样就得到了拖动进度条滑块过程中的拖动进度时间，使用转换格式得到的拖动进度时间在 tipFormatter 函数内格式化 Tooltip 的内容，这样就实现了拖动进度条过程中跟随着拖动的进度显示当前拖动进度对应的具体日期时间。

Slider 组件使用刻度标记 marks 作为标签，在进度条上用刻度标签展示回放过程中的关键帧事件行为。刻度标记是一个包含了刻度位置数组、刻度样式、刻度标签内容数组的 object,它的取值在 slider 组件的[min,max]区间内，通过数据预加载后的关键事件会返回给 历史回放控制流程组件关键事件帧的时间数据与事件代号编码，我们通过在组件生命周期的 componentDidMount 阶段接受到预加载的关键事件帧数据后，使用 moment.js 对事件帧时间数据转换为秒为单位的值作为刻度标记 marks 的刻度位置数组赋值，将事件代号编码数组为刻度内容标签数组进行赋值，并且将刻度标签样式标为红色，这样就实现了进度条关键事件帧结点的渲染。

当回放流程控制组件在组件生命周期的 componentDidMount 阶段完成了数据预加载任务后，将会根据预加载数据初始化 Slider 组件，首先根据回放起止时间经过日期格式转换后初始化 Slider 组件的取值区间与默认取值，然后根据预加载关键事件帧数据，初始化历史回放流程控制组件的刻度标记数组，并且对 Card 组件中的内容信息进行初始化，使用数据自适配的方法，直接更改组件视图中的自适配变量的信息，使得组件自动显示预加载的回放模式、开始时间、停止日期时间、当前回放时间、当前回放速率等信息，最后对于组件内部状态机的状态进行初始化，初始化回放速率状态为正常速率回放状态，初始化缓冲数据时间窗口，最终完成回放流程控制组件的初始化工作[22,23,24]。

我们使用缓冲批量数据加载算法进行历史回放的渲染过程，通过组件内部

的状态机控制历史回放的流程，组件状态机的流程控制状态分为三类，第一类状态为播放状态，该状态使用布尔类型赋值，主要控制当前历史回放是处于回放进行中的状态还是回放暂停状态，还是回放停止状态。组件处于回放进行中状态时，定时器进行使用缓冲批量数据加载算法进行历史回放。回放暂停状态与回放停止状态是不同的，从功能上看，回放暂停后，单击继续播放，仍然可以让回放继续，且回放暂停时刻的数据仍然渲染在了地图上，可以通过暂停回放来具体的查看暂停时刻的回放目标详细信息。而回放停止后，除非重新开始回放，否则历史回放图层会一直清空。第二类状态是速度状态，负责控制当前回放的速率。第三类状态是回放进度状态，主要控制当前回放进度滑块的移动。回放过程中存在的常变量为回放请求间隔、缓冲数据时间窗口、回放速度常量因子。

回放流程控制组件是通过状态机的回放进行中状态与回放暂停状态控制控制缓冲式批量数据渲染，当组件进行初始化后，处于回放暂停状态，此时回放起止按钮的图标由于回放暂停状态，显示为三角形状，当单击回放起止图标后，组件状态机的回放暂停状态转移为回放进行中状态，此状态转移函数将会开启批量数据缓冲定时器与告警定时器，历史回放的告警目标渲染也存在闪烁效果，实现算法与实时告警闪烁渲染相同，同样采用了多定时器渲染的方法，批量数据缓存定时器采用缓冲式数据加载算法进行历史回放目标帧矩阵的逐帧渲染。在播放进行中状态时后回放起止图标变化为暂停图标。

在回放进行中状态单击了暂停图标后，组件状态发生转移，由回放进行中状态转移为暂停态，此时在单击鼠标事件中使用 clearInterval 方法停止批量数据缓冲定时器与告警定时器，由于 React 框架使用虚拟 DOM 模式采用计算 Virtual DOM Diff 的极小值的算法进行组件视图更新，因此仅仅在单击事件内使用 clearInterval 函数无法立即将回放暂停渲染，因为定时器不会马上被清除，需要等待一个组件渲染周期的时间，等待进行虚拟 DOM 计算出视图更新差的特定步骤，为了实现单击暂停图标按钮实现回放立即暂停的效果，我们使用组件生命周期的 componentWillReceiveProps 与 compentDidUpdate 阶段进行 clearInterval 批量数据缓冲定时器与告警定时器停止的操作，因为进行了视图更改操作后会不可避免的进入这两个组件生命阶段，在 componentWillReceiveProps 与 compentDidUpdate 阶段提前对定时器进行停止操作，无需等待虚拟 DOM 装载成为真实 DOM 的过程，由于我们在回放暂停的时候仅仅停止定时器而没有清楚图层，故回放暂停时刻的数据仍然渲染在了地图上，可以通过暂停回放来具体的查看暂停时刻的回放目标详细信息。这样就通过一个回放起止 Button 组件更改组件状态机的方式控制回放的进行和暂停。历史数据回放状态控制流程图如图 4-

26 所示。

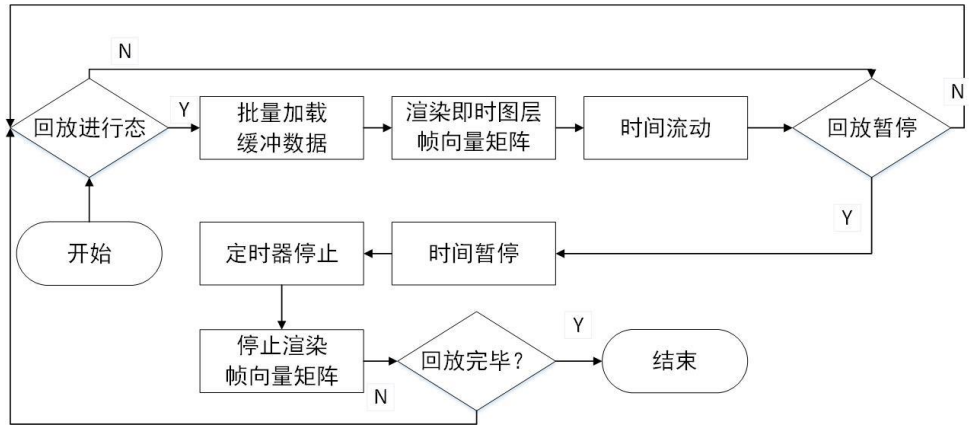


**图 4-26 历史数据回放状态控制流程图**

Slider 组件跟随着缓冲批量数据加载算法渲染回放目标的同时，进度条滑块会依照当前的回放进度进行移动，这是因为每一次缓冲批量数据加载算法进行历史回放帧数据渲染，每一帧的数据在图层上渲染结束后，缓冲数据定时器都会将该时间帧的时间索引封装入信号内，slider 组件通过订阅此时间帧索引数据，依照时间帧索引进行 Slider 当前进度值的更改，这样就实现了 Slider 进度条随着定时器的进度而按照一定的步长滑动。缓冲数据定时器 timer 与当前的进度条 value 本质上都属于流程控制组件的状态，在这里不能通过 timer 去更改进度条的 value 状态，也不能通过组件生命周期去做处理，由于 React 框架的状态更新采 Virtual Dom Diff 算法，故如果通过 timer 更改 sildervalue 状态的话，被更改的状态是无法得到及时的转移，从而就无法实现 Slider 跟随定时器的效果了，这里巧妙的利用了组件的闭环式自回环通信方法，通过组件的 timer 状态组件自己给自己发送闭环信号，强制组件进行即使的组件视图更新的方法，实现了 Slider 跟随定时器的实时滑动。这里表现出了正向的组件状态发送信号引起 Slider 改变，反过来也可以实现 Slider 进行滑动状态改变，发送自回环信号引起组件当前回放帧时间状态改变改变，从而实现定时器 timer 重新使用缓冲批量数据算法缓存批量目标回放数据，实现拖动进度条后，定时器重新缓冲数据，历史数据回放随之依据拖动后的进度进行播放。流程控制组件自回环通信模式如图 4-27 所示。

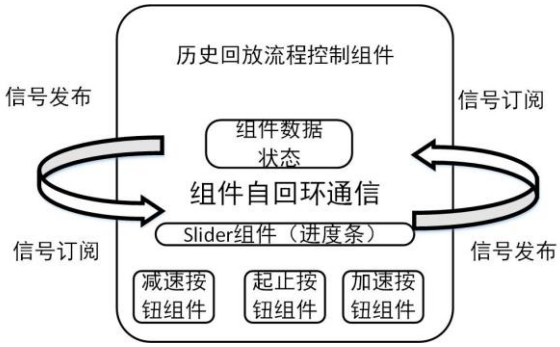


**图 4-27 流程控制组件自回环通信模式**

通过加速或者减速按钮更改当前的回放速率状态，通过组件的自回环通信发布信号给组件内部的批量数据缓冲定时器，定时器收到更改了速度状态信号之后不是马上进行数据的缓冲，定时器会根据当前的速率配合回放的当前时间帧索引判断当前缓存的数据是否可以继续消费，若可以继续消费，证明当前缓存的数据可以继续支撑回放的使用，则继续渲染帧数据，否则进行批量数据重新缓冲。拖动进度条同理，拖动进度条后使用自回环通信发布信号给组件内部的批量数据缓冲定时器，定时器做同样的操作，从而使用加速减速 Button 组件与 Slider 组件实现了进度控制。历史数据回放批量缓冲控制流程图如图 4-28 所示。

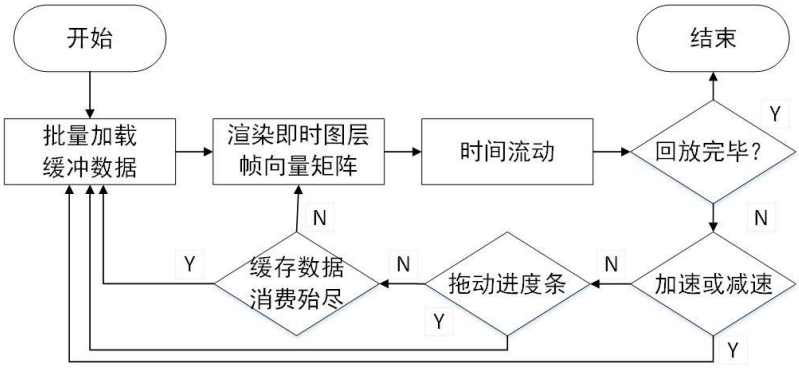


**图 4-28 历史数据回放批量缓冲控制流程图**

数据缓冲等待时，进度条会显示缓冲中字样，此时前端向后台不断请求缓冲数据，并且时间停留在这一刻，不继续走动，数据也停留在这一刻。数据缓冲完成后，即使服务端关闭，本地仍然有缓冲数据，会继续播放缓冲好的数据，直到播放至本地缓冲数据的结束，如果这个时候暂停，也会不影响缓冲数据的播放，且可以加速，实现低耦合的回放。拖动进度条，或者突然减速会重新触发缓冲，并且正常播放时加速，会依据本地缓冲数据先播放缓冲数据中已经有的部分，直到播放至没有的部分，再进行缓冲。加速减速时，拖动进度条会重新触发缓冲且

缓冲时，拖动进度条至已经缓冲好的部分会重新正常播放。历史数据回放渲染流程与速度控制算法如图 4-29 所示。

**Algorithm 6** History Playback Process Render and Speed Control

**Input:**
The state of current playback $S$. The playback timer $T$.
The start time value of history data playback $O$.
The end time value of history data playback $E$.
Play speed factor $r$. The difference of adjacent frame time $D$.

**Output:**
The matrix Batch data $M$.

Create a buffer queue $Q$;
**while** $(S == True)$ **do**
  get current playbackFrameIndex $I := getCurrentFrameIndex()$;
  **if** $(getNextPlackbackFrame(I) == null || isSpeedFactorStateChanged(r) == True || isSlideProcess() == True)$ **then**
    $caculateCurrentSpeed(r)$;
    $Q \leftarrow bufferBatchData(O, E, D, I, r)$;
    $M \leftarrow Q$;
  **else**
    **if** $(I < getEndFrameIndex(E))$ **then**
      $getBatchSupermapMarkerVector(Q, M, I)$;
      $renderFrameData(Q, M, I)$;
      $changeCurrentFrameIndex(I)$;
    **else**
      $stopPlayback()$;
    **end if**
  **end if**
  Return $M$;
**end while**

图 4-29 历史数据回放渲染流程与速度控制算法

#### 4.1.3.4 历史回放事件告警组件实现

历史回放事件告警组件是一个三层嵌套树状组件。快进回放至包含关键帧告警事件、区域重要行为信息的时间节点时会自动暂停弹出告警通知。选择停止快进逐帧播放关键帧事件还是跳过关键帧告警事件。组件包含了 Notification 组件，通知组件中包含了较为复杂的关键帧事件通知内容，并且含有两个 Button 组件，进行带有交互的通知，给出用户下一步的行动点，并且系统进行主动推送。

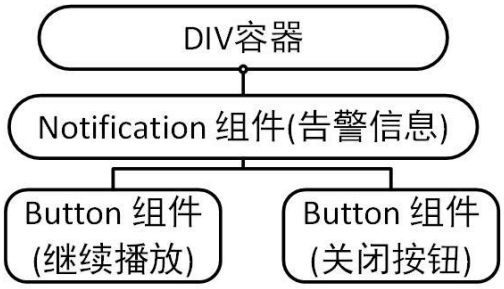


**图 4-30 历史回放事件告警通知组件**

历史回放事件告警通知组件使用 Notification 组件的 open 方法在系统界面的右上角弹出消息通知，并且将历史回放中数据预加载的事件关键帧基本信息显示在 Notification 组件的内容部分，使用单击鼠标事件的方式更改当前历史回放控制流程组件的回放状态，控制其暂停回放，并且清楚缓冲数据定时器，当点击停止快进逐帧播放后，将重新启动定时器，如果此时处于快进状态，则会将速率状态转移为正常速率，进行关键事件逐帧播放。如果还是依然处于快进状态，则会跳过事件。若选择跳过事件，则重新开始播放时候会自动重新启动定时器，并且由于现在处于快进状态，会自动缓冲跳跃当前回放进度状态至关键事件帧之后的数据，快进时处于回放跳越式跃进播放的状态故会自动跳过关键帧事件。快进回放事件告警流程控制如图 4-31 所示。

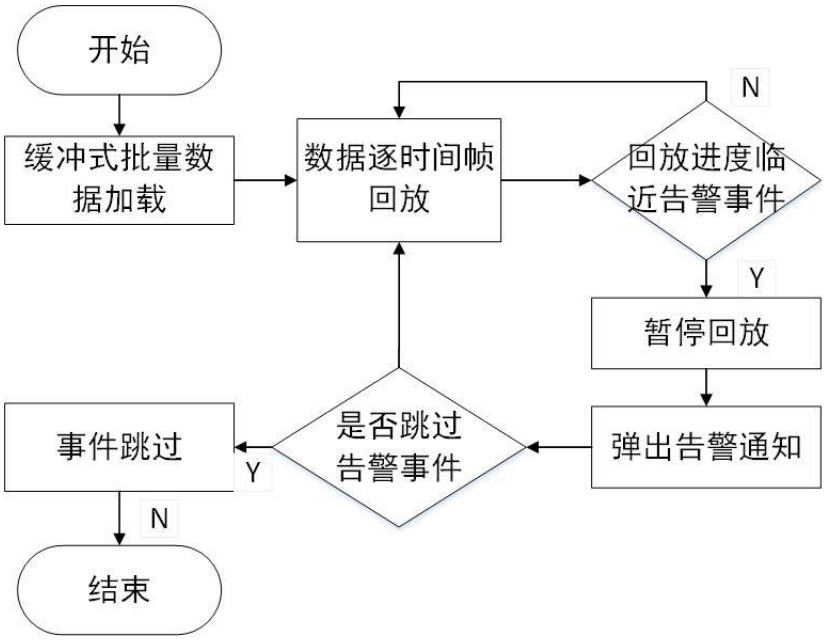


**图 4-31 快进回放事件告警流程控制**

**Algorithm 7** History Playback Warning Behavior and Event Notify

**Input:**
The waring event time array $T$. The state of current playback timer$S$.
The difference of adjacent frame time $D$.

**Output:**
The matrix Warning event data $M$.

```
1: Create a buffer queue Q;
2: while (S == True) do
3:    get current playbackFrameIndex I := getCurrentFrameIndex();
4:    while (isArrayEmpty(T) == False) do
5:       get current event time Index T_c=getCurrentEventTimeIdex(T);
6:       if (I < T_c&&I + D > T_c) then
7:          caculateCurrentSpeed(r);
8:          M ← getEventInfo(T_c);
9:          publishSignaltoWarningNotification(M);
10:         pauseHistoryPlaybackProcess();
11:         Return M;
12:      end if
13:      changeCurrentEventTimeIdex(T_c);
14:   end while
15:   continueHistoryPlaybackProcess(Q, M, I);
16: end while
```

**图 4-32 历史批量缓冲数据回放事件告警通知算法**

历史回放告警事件，快进播放至下一个回放时间就是出现告警的事件的进度时，即这一刻还未报警，下一刻出现了报警事件会自动暂停，弹出通知框并暂停。故在进行快进回放时候，定时器会不断的将当前时间帧索引与与预加载的关键事件事件帧矩阵进行匹配，如果匹配符合快进忽略关键事件帧的条件则弹出告警通知。历史批量缓冲数据回放事件告警通知算法如图 4-32 所示。

## 4.2 态势呈现系统空间范式设计与组件化实现

### 4.2.1 目标跟踪判读设计与组件化实现

#### 4.2.1.1 目标跟踪视觉与判读效能模型

在目标视觉跟踪中，传统的基于相关滤波器的方法都存在功能冗余和缺乏运动信息的瓶颈。我们可以采用新的跟踪框架称为多层次独立相关滤波器[25]1 进行目标视觉对象跟踪。它的框架体系结构如下图所示。

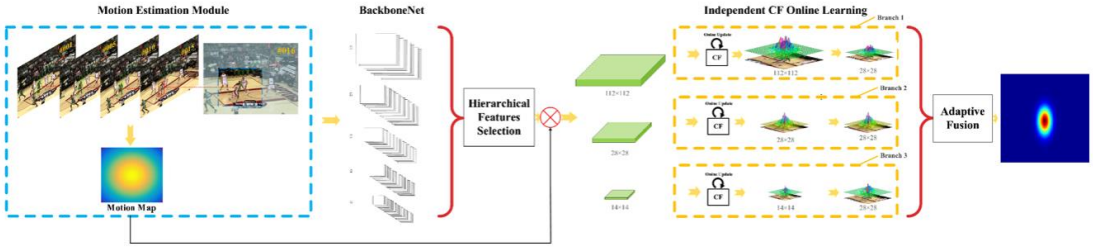


**图 4-33 MHIT 框架体系**[25]2

多层次独立相关滤波模型为每个特征训练一组独立的滤波器[25]4：

$$x_l = \sum_{i}^{D_l} x_{l,i} \tag{4-39}$$

其中 Xl,i 是 lth 层和 ith 通道的特性。卷积神经网络的每一层都可以看作是一组非线性滤波器。随着特征尺寸的增加，将在算法中引入更多的复杂性和冗余。因此，输入图像由每层的滤波器编码。如果一层中有 D 滤波器且特征映射的大小为 M 乘 M，则对应于特征映射的通道数也是 D。随着层次特征的加深，D 逐渐增加，所获得的特征映射的复杂度也逐渐增加。如果我们直接组合多维特征，特征的维度将会很高，这会导致计算负担大大增加[25]3。

因此，通过增加所选级别或层数的特定区域来提高结果的准确性是有效的。相反，太多的层将导致选择正确的结果更加困难并遇到许多计算的负担。因此，最好的解决方案是妥协特定区域和层数。分解的目标函数可以表示为 L 个独立的解决方案目标函数[25]5：

$$\arg\min \sum_{k=1}^{K} ||\phi(x_{l,k}, f_l) - y_{l,k}||_{L^2}^2 + \lambda \sum_{d=1}^{D_l} ||w * f_{l,d}||_{L^2}^2 \tag{4-40}$$ [25]5

其中 fl,d 是 lth 层和 dth 通道滤波器参数，yk 表示预定义的高斯窗口目标函数[25]4。

对于方程 14 中的上述优化问题，我们首先求解滤波器参数 f。对于每组滤波器，将导数设置为零，并且通过以下正规方程求解方程的最小值[25]5，其中

$$\Gamma = diag(f_{l,1}, f_{l,2}, ..., f_{l,D_l}) \in R^{D_l \times D_l} \tag{4-41}$$ [25]5

$$Af = \Gamma^T X y \tag{4-42}$$ [25]5

$$A = \Gamma^T X X^T \Gamma - \lambda W^T W \tag{4-43}$$ [25]5

对于多个独立分支，分层过滤器得到解决，使用一种自适应权重方案，以有效地融合并获得更稳健的结果。最终损失函数表述如下[25]5：

$$E(f,m)=\sum_{l=1}^{L} m_l E_l(f)+\sum_{l=1}^{L}||m_l||_{L^2}^2 \quad s.t.\ \sum_{l=1}^{L} m_l=1, m_l>=0 \tag{4-44}$$

将每个层的结果表示为 E。m 的优化问题可以转换为[25]5：

$$\arg\min_{m} \mathbf{m^T E}+\mathbf{m^T m}, \quad s.t.\ \sum_{l=1}^{L} m_l=1, m_l>=0 \tag{4-45}$$

下图展示 MHIT 模型了与前十大最佳跟踪器的预期平均重叠，鲁棒性和准确度的比较。可以看到 MHIT 在鲁棒性，平均重叠，预期平均重叠方面都优于其他模型[25]6。

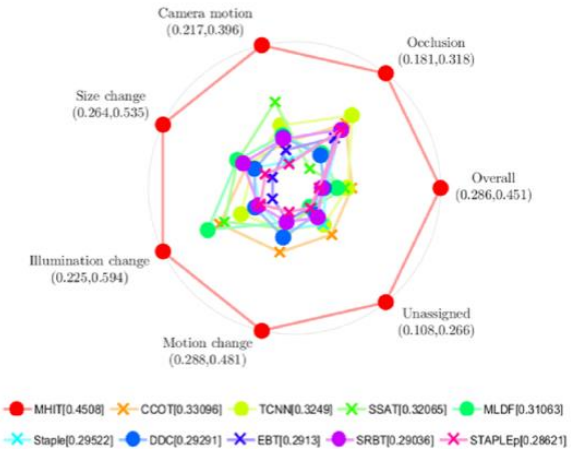


**图 4-34 MHIT 比较图**[25]6

实时与历史结合的态势系统实际上是一个典型的多源传感器信息融合系统，针对海洋环境的复杂性和手段自身特点运用层次分析法原理来构建探测手段的效能模型。探测能力是评价多手段协同探测效能的主要指标，它反应了多手段协同完成可能赋予的探测任务的能力度量，可以用目标探测能力、目标定位能力、目标追踪能力、识别能力和目标容量来进行综合评价。

目标判读效能模型的整个体系的探测威力区是体系中所有当个手段威力区的合成。不同的目标高度、不同的目标截面积、以及不同的发现概率下体系探测威力区大小是不同的，如果受到干扰，体系的探测威力区会发生明显的变化。目标判读效能模型的探测能力评估指标有：覆盖系数、重叠系数、干扰压制比。

1) 覆盖系数定义为监控区内体系的探测威力区面积和责任区面积之比：

$$C_{ov} = \frac{Y_{i=1}^{n}(A^{i}IA^{0})}{A^{0}} \tag{4-46}$$

式中，$A^{i}$表示第 i 个监控探测手段的探测威力区面积，$A^{0}$是责任区面积。所以，威力区覆盖系数最小为 0，对应于监控探测体系探测威力区与责任区不想交；若有部分相交，则覆盖系数大于零，覆盖系数越大，说明监测探测体现的探测空间越能满足监控需求。

2)重叠系数是指监控探测手段探测威力区面积之和与真个体系探测威力面积之比：

$$C_{cd} = \frac{\sum_{i=1}^{n} A^{i}}{Y_{i=1}^{n}(A^{i}IA^{0})} \tag{4-47}$$

3)干扰压制比

监控探测体系由于受到干扰使探测威力面积减小。干扰压制比定义为监测探测体系受到干扰后探测威力区面积与未受干扰比时的探测威力区面积之比：

$$J = \frac{Y_{i=1}^{n} A^{J,i}}{Y_{i=1}^{n}(A^{i}IA^{0})} \tag{4-48}$$

$A^{J,i}$表示干扰后体系中第 i 个监控探测手段的探测威力区面积。

综合考虑上述评价指标，最后得到多手段协同探测体系的探测能力计算模型：

$$C_{系统} = （1 + C_{ov}）^{k_1} \circ C_{cd}{}^{k_2} \circ (1 + J)^{k_3} \tag{4-49}$$

$k_1$、$k_2$、$k_3$表示各系数在监控探测体系探测能力中所占的权重因子，且 $\sum_{i=1}^{3} k_i = 1$。

目标判读效能模型目标定位能力是监控探测体系对目标的定位是在探测发现的前提下融合体系内其他手段的探测的观测数据，运用一定的定位算法，给出目标的空间位置。一般对定位能力的评估有专家评估法，定性分析和模糊综合评定法等，或者运用探测手段传感器的技术参数来进行计算。这些方法使得评估结果缺乏客观性或者不能体现海洋环境所带来的干扰。为了定量评定海洋环境下多手段协同探测体系对目标定位的好坏采用定位精度几何稀释（GDOP）作为衡量其定位性能的指标：

$$GDOP = \sqrt{e_x^2 + e_y^2 + e_z^2} \tag{4-50}$$

$e_x^2$ 、$e_y^2$ 、$e_z^2$代表定位三维坐标的定位误差的均方差。

目标判读效能模型的目标跟踪能力可以用目标航迹和光电目标追踪精度来表示多手段协同探测体系的目标追踪能力。

$$E_{总} = E^{i}_{雷达航迹} + E^{i}_{光电} \tag{4-51}$$

I 代表某种传感器手段的个数。E 代表归一化后某种传感器的跟踪精度。

目标判读效能模型目标识别能力目标识别是指对目标类型和属性进行正确识别的评价，包括目标类型识别概率和目标属性识别概率。定义阶跃函数：

$$W（X,Y）\begin{cases}1 & X = Y\\ 0 & X \neq Y\end{cases} \tag{4-52}$$

X 为识别的目标，Y 表示真实的目标。单目标识别率可以定义为：

$$P = \frac{1}{N}\sum_{i=1}^{N} W(X,Y) \tag{4-53}$$

N 代表目标融合后不同位置的个数，进而得到多目标识别精度。定义目标的个数为 K 个，则对目标属性和类型融合后的综合正确判断概率为：

$$P = \frac{\sum_{j=1}^{K}(N_j p_j)}{\sum_{j=1}^{K} N_j} \tag{4-54}$$

$p_j$代表单个目标类型和属性融合的正确识别的概率。

目标判读效能模型的目标容量是指最大处理的监控区目标数目。一般来说，当目标容量小于等于监控区域可能出现的最大威胁目标数目的时候，它是越大越由型的指标；当大于监控区域可能出现的最大威胁目标数量的时候，再增加体系目标容量对体系的效能也没有明显的影响。

#### 4.2.1.2 迁徙态势组件实现

迁徙态势组件包括了 Model 组件，内部容纳了两个 Button 组件进行操作交互，使用 Transfer 组件选择需要查看迁徙转移态势的目标，使用 Ranger 组件选择查询目标空间位置数据的时间段。组件实现了特定单个目标在特定时段内的空间历史轨迹经过数据采样、关键轨迹帧提取、历史轨迹查询后展示出目标的空间迁徙转移态势，并展示出目标迁徙轨迹的目标历史静态时空信息。目标迁徙过程中的发生的事件、区域行为、告警信息，使用高密度、可伸缩的图表渲染在可视化界面之上。迁徙态势组件结构如图 4-35 所示。

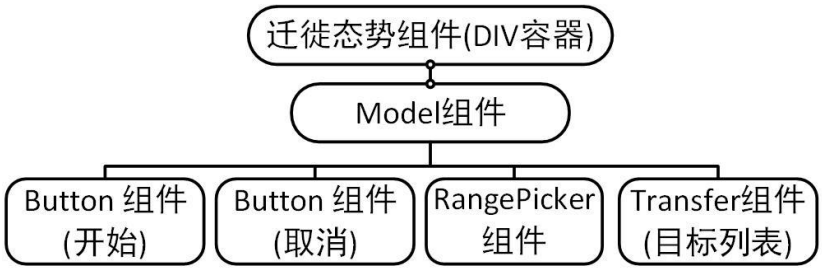


**图 4-35 迁徙态势组件结构**

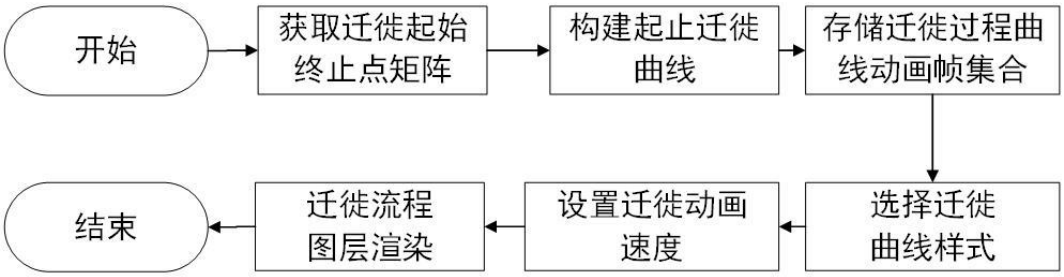


**图 4-36 迁徙态势渲染流程**

迁徙态势渲染流程如图 4-36 所示。迁徙是基于空间的、全局范围内的粗粒度的展示，单击迁徙态势后将目标在某一时期内的迁徙轨迹展现出来，并且每一条轨迹的终点都有一个标记，单击标记会弹出这条轨迹的历史信息。单击后应该选择目标，并选择目标跟踪判读的开始结束时间，后台返回，目标迁徙的采样起止坐标点。可以是单目标，也可以是多目标显示时，不同目标的迁徙轨迹之间应该颜色不同，并且在图表上事件应该用不同的颜色，并且表明图例。单击迁徙态势后，自动向后台发送请求，将当前的目标表数据展示在列表中，单击提交后，将要跟踪判读的目标 ID 号发送给后台，后台返回这一段时间内历史的迁徙起止点，在 AJAX success 函数中根据得到的 data，拟合出迁徙曲线，并且将迁徙的信息写入弹出框内，并且最终将迁徙曲线与动画效果展示在 mapv 图层上，注意这里拖动地图图层要对 MapV 图层进行重新绘制，否则会产生动画错误。

目标迁徙过程中的事件、行为（进出防护区）、告警信息等，用一个 scatter 图表显示，并且能够展示具体的告警事件、地点、时间、特征信息，及进出防护区的事件告警信息。内置型数据区域缩放组件可以概览数据整体，按需关注数据细节。展示了迁徙轨迹后，应该将事件表格也一并初始化，包括初始化图例，初始化不同图例的不同数据。再次单击迁徙态势，取消当前的所有展示内容，清除图层，并且将图表数据清除，并且将图表设置为不可见。

**Algorithm 8** Target Migrate Curve Render

**Input:**

The start point array of migration $O$.
The end point array of migration $E$.
The mapv curve render layder $L$.

**Output:**

The render migration curve set $M$.

1: **for all** $i$ such that $0 \leq i \leq sizeof(O)$ **do**
2: $lineData[i] := pushPointData(O[i], E[i])$;
3: $curveData[i] := mapvUtilCurve(lineData[i])$;
4: **for** $j = 0$ to $sizeof(curveData[i])$ **do**
5: $timeMoveData[i][j] := pushTimeMovePoint(curveData[i][j])$;
6: **end for**
7: **end for**
8: $M := pushData(curveData, timeMoveData)$;
9: $setCurveFillStyleShadowColor(L)$;;
10: $L \leftarrow addDataSettoLayer(M)$;

**图 4-37 迁徙态势渲染算法**

迁徙态势渲染算法如图 4-37 所示，首先进行特定时段内的历史轨迹查询，并且得到关键轨迹起始点矩阵，然后对关键轨迹起始点矩阵进行遍历，为每一个起止点构建起止迁徙曲线，每建立了一条起止迁徙曲线，都要对这条迁徙曲线进行曲线点遍历采样形成迁徙过程曲线动画帧点集合，最终选择迁徙曲线样式，设置迁徙动画速度，使用 Mapv 框架中的 Supermap.Mapv 图层对上述的迁徙数据集进行图层渲染。

#### 4.2.1.3 热点区域组件实现

热点区域组件是一个三层结构，针对单目标经常出现的区域，系统以热力图的形式，通过色带渲染目标空间位置数据疏密程度及区域访问频度，通过核密度分析法使用缓冲区交叉填充灰度叠加，然后进行色带空间灰度信息映射，从而用冷色到暖色展示出目标热力点相对密度与加权密度状态。内部容纳了两个 Button 组件进行操作交互，使用 Transfer 组件选择需要查看热点区域的目标，使用 Ranger 组件选择查询目标空间位置数据的时间段。热点区域组件结构如图 4-38 所示。

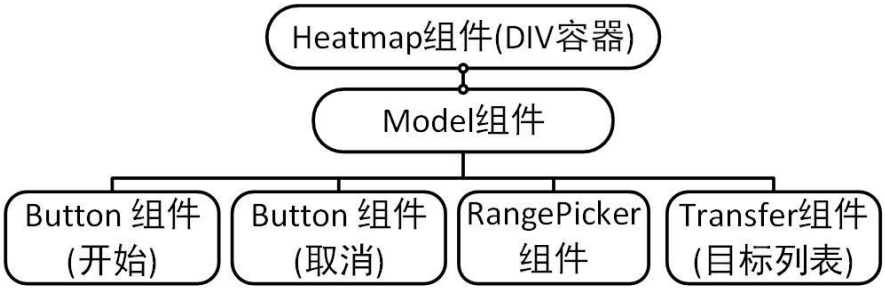


**图 4-38 热点区域组件结构**

热点区域渲染流程如图 4-39 所示。单目标经常出现的区域以热点图的形式显示出来，还可以显示目标经常出没的区域具体是在什么时间点经常出没。

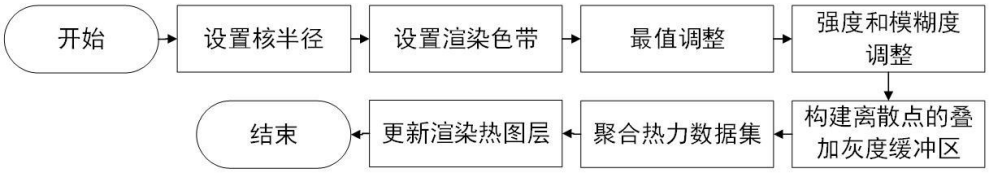


**图 4-39 热点区域渲染流程**

热点区域渲染算法如图 4-40 所示。首先对于设置核半径，然后设置一个表示了最终热力图效果的渲染色带，它是通过空间灰度值映射而来的。然后我们调整色带的最大值和最小值，并且对与其模糊度与强度等渲染效果进行调整，然后使用以上参数，对于空间查询出的目标位置数据集建立缓冲区，然后经过缓冲区的叠加得到了灰度值，将灰度值进行映射至色带，再在热力图层上进行渲染就完成了热点区域渲染流程。

**Algorithm 9** Target Region Heatmap Render

**Input:**

The matrix of point $P$. The Heatmap render layder $L$.

**Output:**

The render Heatmap set $M$.

1: $setNuclearRadius(L)$;
2: $setGrayBuffer(L)$;
3: $setPointsWeight(P)$;
4: $adjustRenderRibbon(L)$;
5: $ajustIntensity(L)$;
6: $adjustMaximunAmbiguity(L)$;
7: $L \leftarrow addDataSettoLayer(P)$;
8: $M \leftarrow renderHeatmapLayer(L)$;

**图 4-40 热点区域渲染算法**

#### 4.2.1.4 数据蜂窝组件实现

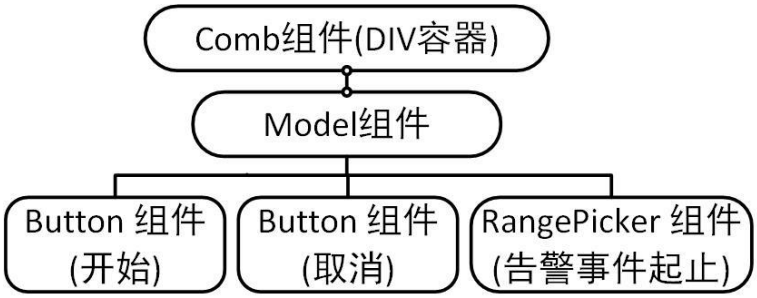


**图 4-41 数据蜂窝组件结构**

数据蜂窝组件结构如图 4-41 所示，组件分为三层，使用 RangePicker 组件选定告警时间的起止时间，使用网格聚合图的方式展现出目标告警事件、行为的空间数据的分布特征和统计特征。基于网格聚合算法，将空间区域划分为一个嵌套的、层次化的网格单元集合，每个网格单元都具有目标告警事件的统计信息。

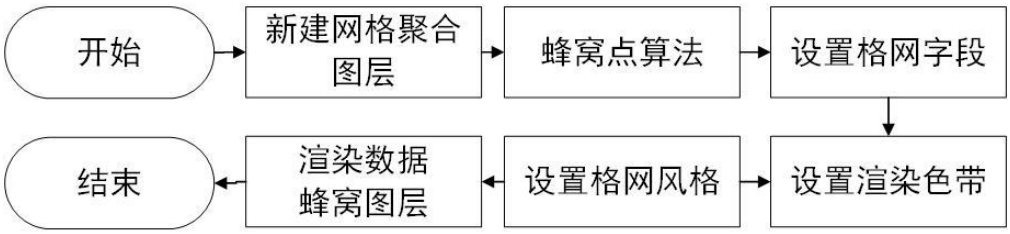


**图 4-42 数据蜂窝渲染流程**

防护区事件、热点事件多发的区域，应该有自己的独特的特征，这种区域可以用数据蜂窝的方法展示，数据蜂窝渲染流程如图 4-42 所示。

网格聚合图简单说就是一种使用空间聚合方法，表现空间数据的分布特征和统计特征。它的基本原理是基于网格聚合算法，将空间区域划分基于网格聚合算法，将空间区域划分为一个嵌套的、层次化的网格单元集合，每个网格单元都具有统计信息。

MapV 框架支持对空间点数据构建网格聚合图，并且提供了两种形状的网格进行聚合显示，一种是矩形网格，一种是六边形网格。通过网格对地图点要素进行网格划分，然后，计算每个网格单元内点要素的数量，并作为网格的统计值，也可以引入点的权重信息，考虑网格单元内点的加权值作为网格的统计值；最后

基于网格单元的统计值，按照统计值大小排序的结果，通过色带对网格单元进行色彩填充。

进行数据蜂窝渲染首先新建网格聚合图层，使用六边形网格单元作为为一致大小的格网，并且地图比例尺的变化，网格单元的大小固定不变；网格用来统计落入每个网格单元内的点对象数目。每个网格中心具有一个标签，该标签为网格单元的统计值，该统计值可以是落入每个网格单元内的点对象数目，也可以是落入每个网格单元内的点的加权值。网格单元的填充颜色表示网格统计值的分布趋势，其颜色由深到浅，表示网格单元的值从大到小。另外，还可以设置网格矩形边框的风格。

**Algorithm 10** Honeycomb point and region render

**Input:**

The matrix of point $P$. The Honeycomb render layder $L$.

**Output:**

The render comb set $M$.

$dx := 3 * sideLength(P)$; $dy = sideLength(P)/2 * 1.73$; $indent := dx/2$;
$xmin := XMin(P) - dx$; $ymin := YMin(P) - dy * 3.0$;
$xmax := XMax(P) + dx$; $ymax := YMax(P) + dy * 3.0$;
$numrows := int((ymax - ymin)/dy) + 1$;
$numcols := int((xmax - xmin)/dx) + 2$;
$y := ymin$;
**for** $r = 0$ to $range(numrows)$ **do**
  $x := xmin - indent/2$;
  **if** $(r\%2! = 0)$ **then**
    $x+ = indent$;
  **end if**
**end for**
**for** $c = 0$ to $range(numcols)$ **do**
  $p := Point(P)$;
  $p.X := x$; $p.Y := y$;
  $x+ = dx$; $y+ = dy$;
**end for**
$setGridAggregationRenderRibbon(L, P)$;
$L \leftarrow setGridFeildFrameStyle(L, P)$;
$M \leftarrow renderCombLayer(P)$;

**图 4-43 数据蜂窝渲染算法**

据蜂窝渲染算法如图 4-43 所示。首先使用蜂窝点算法生成蜂窝六边形，然后设置格网字段，指定了一个格网字段，那么该字段值将作为点的权重信息，此时，网格聚合图每个格网单元的统计值为落在该单元格内的点的加权值。另外，所指定格网字段为数值型。设置渲染色带，通过最大值颜色和最小值颜色渲染网格聚合图中统计值最大与最小的格网单元，其他格网单元使用渐变色带中的其他

颜色渲染，并遵循统计值越大渲染颜色越靠近色带中的最大值颜色一端。设置格网的风格，包括了格网大小、格网边框、格网标签。通过指定边长来确定大小，格单元矩形边框线的线型，有三种情况：无边框、实线边框、虚线边框，还需要设置边框线宽度、边框线颜色、半透明效果。设置网格单元内统计值标签的风格。

### 4.2.2 目标画像设计与实现

目标画像服务是将目标行为、特征、事件的差异，将他们区分为不同的类型，从每种类型中抽取典型的特征赋予标识、统计要素等描述，形成一个目标原型，即目标信息标签化，并且对这些特征分析统计挖掘潜在价值信息，建立目标数据立方，抽象出一个目标的特征全貌。目标画像能够帮助安防监控者找到对的真正需要关注的目标，分析统计挖掘潜在价值信息从而能够一定程度上预测这个目标下一步的行动。

首先将目标信息标签化，系统通过一系列算法或规则挖掘得到或者根据行为数据直接得到目标特征信息，并且将其标签化，特征是有一定的时效性或者半衰期的，针对不同时间得到的标签需要进行权重加权处理。比如时间越近，权重越高。目标的感知特征标签包括国籍、长宽、船类型、目的地、旋转速率、径向速度、电磁频率、实际航向等，目标行为属性标签包括进出防护区情况、告警情况、出没区域情况（以经纬度形式呈现）、船只状态事件类型（地区问题、人员落水、速度异常、转向过大、持续靠近岛屿、尺寸异常、状态异常、船名异常）、防护区停留时间、靠近岛屿最近距离等。

然后构建目标原型数据立方，目标画像不仅仅是将目标的行为特征抽取出来赋予目标一个类型或者原型，它还能展示这个原型本身的信息，即我们建立的情报库中关于这一类的目标的一些历史行为、威胁告警数据，这就实现了对这些特征建立数据立方，分析统计挖掘目标原型的潜在价值信息从而能够一定程度上预测这个目标下一步的行动。假设我们实现了电磁管控，某一个目标被打上了超高频通信的标签，但是它没有触发任何的告警事件，也暂时没有产生危险行为，但是这一类高频通信的历史目标中大部分都是威胁等级高的目标，可以对当前目标做一定程度的预测。

要准确识别目标，依据现有的数据关联方案基本可以做到从设备、区域、路径三个方面对目标进行目标动态行为分析及轨迹跟踪。我们需要结合四大维度时间、地点、目标静态数据、威胁或者行为去评估目标。

本系统对目标画像做了初步的尝试，实现了目标原型的初步画像，画像方法

如下：

1） 拥有相似特征的目标（长宽比 平均速度）占总目标的百分比及其按照告警等级的百分比。

2） 选择目标长宽比特征,与威胁程度指标进行聚合分析聚合分析

在使用 Echart 与组件化技术进行状态管理，在 componentDidUpdate 中渲染原型视图，通过在 activeKey 状态转变为原型视图后再渲染原型视图的方法解决了渲染原型视图渲染问题。具体画像效果如下图 4-44，4-45 所示。

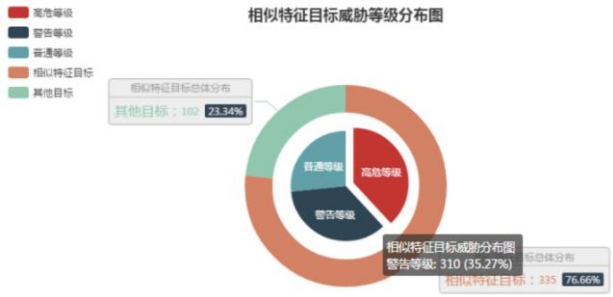


图 4-44 相似特征目标威胁等级分布图

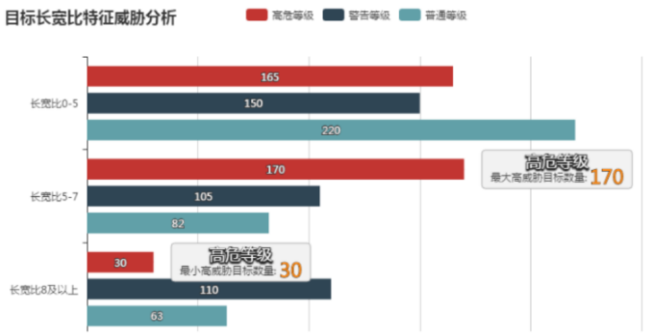


图 4-45 目标长宽比特征数据立方

## 4.3 本章小结

本章详述了态势呈现系统的时空范式设计与实现的组件化实现方式。系统采用时间范式设计了实时态势监控模式与历史数据回放模式，采用空间范式设计了目标跟踪判读与目标画像服务。本章详细介绍了时间范式中的实时与历史结合的双展示模式在系统中的组件化实现方式，以及实现它们蕴含的功能进行的算法、流程设计。详细的介绍了使用空间范式，从全局进行粗粒度的空间历史回放，介

绍了空间大数据可视化的具体实现方法。

# 第五章 系统测试与验证

本章对系统的功能与性能进行测试。

## 5.1 系统测试目标与硬件测试环境

### 5.1.1 系统测试目标

本文设计实现了实时与历史结合的多视图目标态势呈现系统。系统基于目标信息图谱实现了实时态势监控与历史数据回放，并且基于 GIS 服务实现了跟踪判读与目标画像，采用时空范式设计并实现了支持实时历史双展示模式的态势呈现系统。从时间维度对目标的多维度数据视图进行实时监控与历史回放，从空间维度对于目标的历史存储空间位置信息进行空间大数据可视化。本章对系统的功能与性能进行详细测试与验证，通过搭建系统测试仿真平台环境并描述每个功能的测试方法、功能测试流程、测试结果，从而证明本系统是一个具备展示信息关联性，深度认知目标、遵循时间与空间范式并具备实时历史双展示模式的态势呈现系统

系统进行测试的目的就是验证态势系统功能的有效性与可靠性。测试的功能为：实时监控模式下目标信息图谱以及原始感知、特征、行为事件、威胁判据多维度数据视图的展示，实时监控模式的设备显控联动、视频监控、实时告警、情景回放功能，历史数据回放模式的历史回放展示方式、缓冲式批量数据加载算法、历史回放流程控制、历史回放事件告警，跟踪判读服务的迁徙态势、热点区域、数据蜂窝，还有目标画像功能。系统通过功能测试来验证态势系统功能的有效性与可靠性。

我们针对系统的响应时间、并发缓冲性能、稳定性能分别进行测试，并进行定量分析，最终定量分析的结果验证了系统具备可靠有效性。

### 5.1.2 硬件测试环境

（1）平台结构

测试平台结构图如图 5-1 所示。

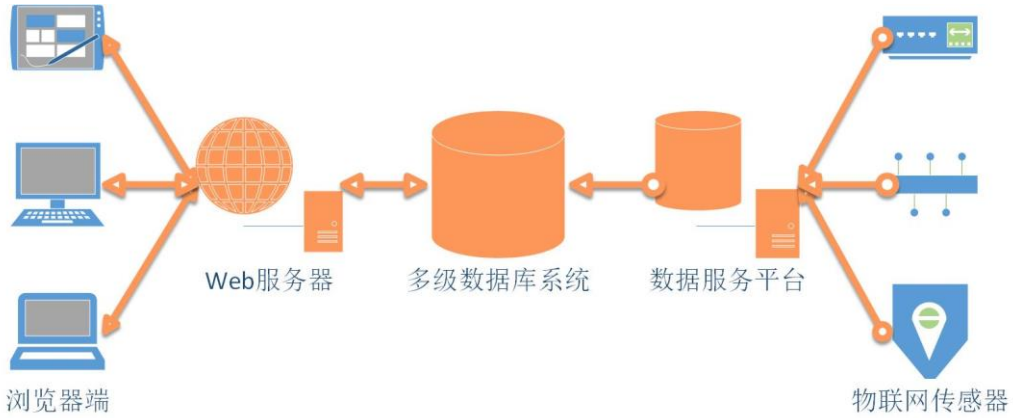


**图 5-1 系统环境拓扑结构图**

系统仿真测试平台拓扑结构包括：

客户端：系统 Browser 端，负责进行可视化。

服务端：服务端使用 tomcat 6.0.45 轻量级 Web 容器，提供服务端的后台数据接口服务、控制命令发布订阅服务。

多级数据库系统：部署在服务器上的 Neo4j 图数据库、HBase 列式数据库与 MySql 关系型数据库，存储多维度数据视图与目标信息图谱。

数据服务平台：提供 IoT 接入服务、实时计算与历史挖掘服务、流媒体服务、GIS 服务等计算密集型服务。

物联网传感器：各种底层传感器设备，实现多源异构的原始感知数据的采集、数据报上传等功能。

（2）仿真测试平台硬件

本章中搭建的系统测试硬件环境为

**表 5-1 测试机器硬件参数**

| | |
|---|---|
| **操作系统** | Windows 7 |
| **处理器类型** | 英特尔 i7 处理器 2.5GHz 主频 |
| **内存** | 32.00GB |
| **系统位数** | 64 位操作系统 |

## 5.2 系统功能测试

本节对系统所有功能进行测试，包括了实时监控模式下目标信息图谱以及

原始感知、特征、行为事件、威胁判据多维度数据视图的展示，实时监控模式的设备显控联动、视频监控、实时告警、情景回放功能，历史数据回放模式的历史回放展示方式、缓冲式批量数据加载算法、历史回放流程控制、历史回放事件告警，跟踪判读服务的迁徙态势、热点区域、数据蜂窝，还有目标画像功能。

### 5.2.1 实时态势监控模式测试

#### 5.2.1.1 原始感知数据视图测试

对于实时监控模式原始感知数据视图的显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-2 所示：

**表 5-2 实时监控模式原始感知数据视图测试**

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件。<br>（3） 激活实时监控组件后，组件变红，在地图上出现实时监控的目标标记点，单击目标标记，打开多维度数据视图卡片容器组件。<br>（4） 鼠标单击多维度视图卡片容器的多维度数据标签，打开原始感知数据视图。 |

测试结果显示，系统能够实时的正确获取原始感知数据视图的信息，并且进行可靠的实时渲染，实时渲染效果流畅无卡顿，原始感知数据视图完整的展示出了目标多维度感知数据，并且界面操作逻辑正确、流畅、无卡顿。测试结果如图 5-2,5-3 所示。


图 5-2 原始感知数据视图整体可视化效果


图 5-3 原始感知数据视图卡片容器可视化效果

### 5.2.1.2 特征数据视图与目标信息图谱测试

对于实时监控模式特征视图及目标信息图谱的显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-3 所示：

表 5-3 实时监控模式特征视图及目标信息图谱测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件。<br>（3） 激活实时监控组件后，组件变红，在地图上出现实时监控的目标标记点，单击目标标记，打开多维度数据视图卡片容器组件。<br>（4） 鼠标单击多维度视图卡片容器的特征视图标签，打开特征视图与目标信息图谱可视化 |

测试结果显示，系统能够实时的正确获取特征视图及目标信息图谱的信息，并且进行可靠的实时渲染，实时渲染效果流畅无卡顿，特征视图及目标信息图谱完整的展示出了目标特征数据以及目标信息图谱，并且界面操作逻辑正确、流畅、无卡顿。测试结果如图 5-4,5-5 所示。

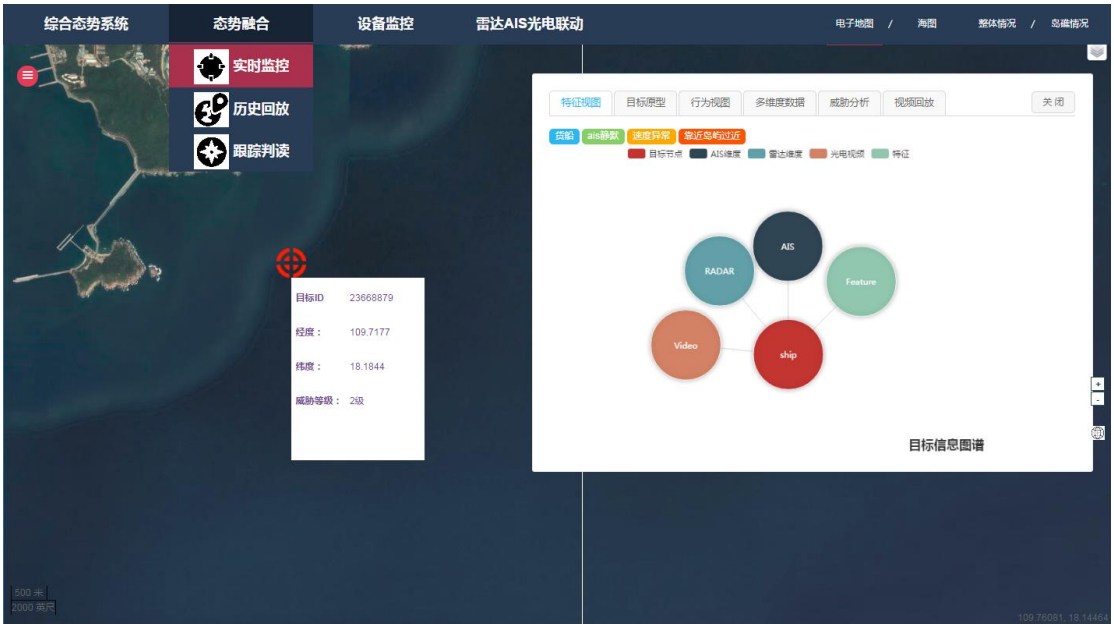


图 5-4 特征数据视图及目标信息图谱整体可视化效果


**图 5-5 原始感知数据视图卡片容器可视化效果**

#### 5.2.1.3 行为事件数据视图测试

对于实时监控模式特征视图及目标信息图谱的显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-4 所示：

**表 5-4 实时监控模式行为事件数据视图测试**

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件。<br>（3） 激活实时监控组件后，组件变红，在地图上出现实时监控的目标标记点，单击目标标记，打开多维度数据视图卡片容器组件。<br>（4） 鼠标单击多维度视图卡片容器的行为视图标签，打开行为事件数据数据视图 |

测试结果显示，系统能够实时的正确获取行为事件数据数据视图的信息，并且进行可靠的实时渲染，实时渲染效果流畅无卡顿，行为事件数据数据视图使用结构化的表格完整的展示出了行为事件数据数据，并且界面操作逻辑正确、流畅、无卡顿。测试结果如图 5-6,5-7 所示。

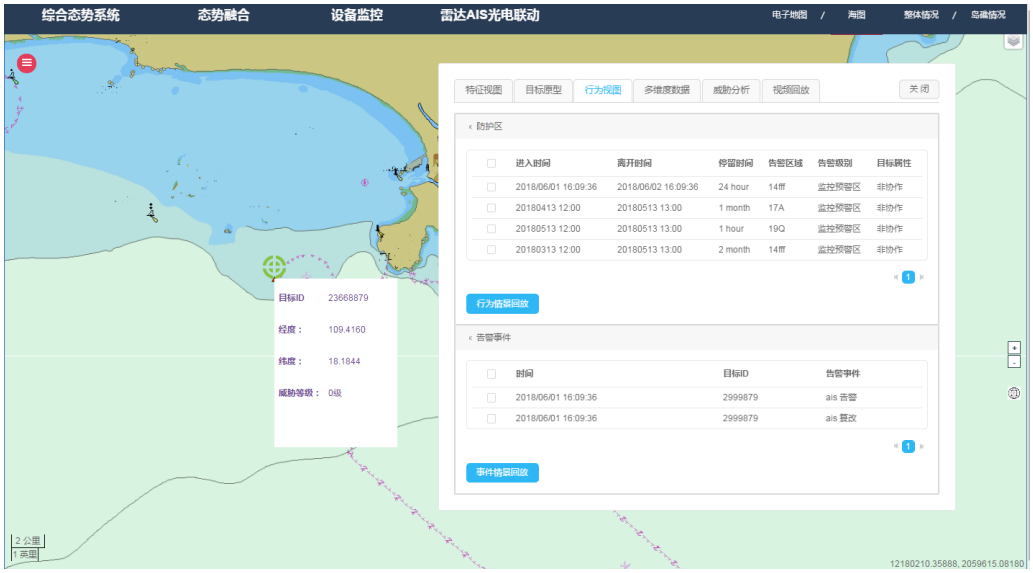


图 5-6 行为事件数据视图整体可视化效果

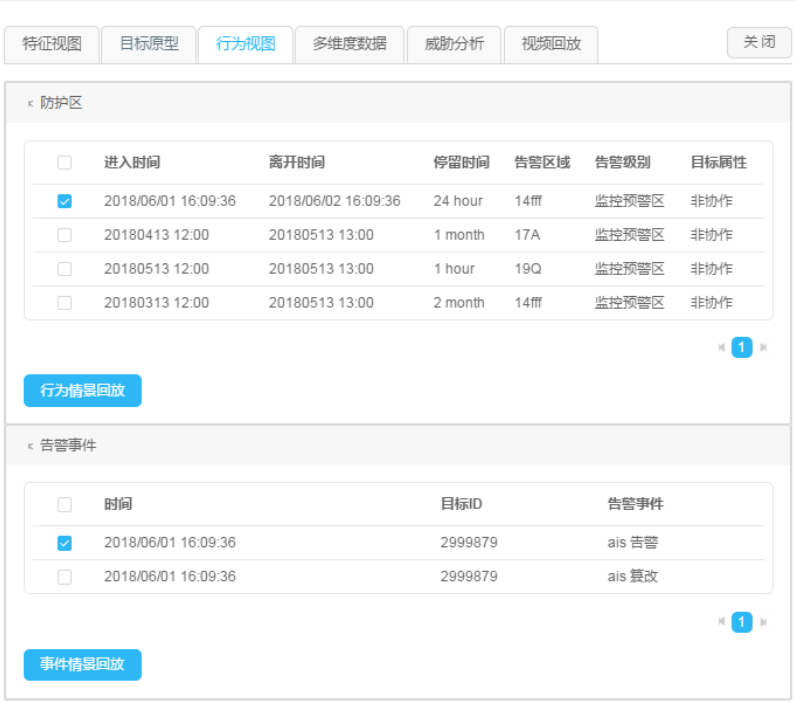


图 5-7 行为事件数据视图卡片容器可视化效果

### 5.2.1.4 威胁判据数据视图测试

对于实时监控模式威胁判据数据视图的显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-5 所示：

表 5-5 实时监控模式威胁判据数据视图测试

| | |
|---|---|
| 功能测试流程 | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件。<br>（3） 激活实时监控组件后，组件变红，在地图上出现实时监控的目标标记点，单击目标标记，打开多维度数据视图卡片容器组件。<br>（4） 鼠标单击多维度视图卡片容器的威胁分析标签，打开威胁判据数据视图 |

测试结果显示，系统能够实时的正确获取威胁判据数据视图的信息，并且进行可靠的实时渲染，实时渲染效果流畅无卡顿，威胁判据数据视图使用完整的展示出了目标威胁等级与威胁判据以及告警跃迁的过程，并且界面操作逻辑正确、流畅、无卡顿。测试结果如图 5-8,5-9 所示。


图 5-8 威胁判据数据视图整体可视化效果


图 5-9 威胁判据数据视图卡片容器

#### 5.2.1.5 设备显控联动与视频监控跟踪测试

对于实时监控模式设备显控联动与视频监控跟踪显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-6 所示：

**表 5-6 实时监控模式设备显控联动与视频监控跟踪测试**

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区设备显控标签，自动弹出下拉组件列表，依次单击下拉列表内的各个设备组件，开启设备显控功能。并且开始实时监控模式。<br>（3） 激活设备显控功能后，组件变红，在地图上出现实时监控的目标标记点与设备感知信息，在目标标记点或者设备标记点右键单击，弹出菜单，鼠标单击菜单项光电随动启动，弹出光电视频框，完成设备联动功能。<br>（4） 单击目标标记，打开多维度数据视图卡片容器组件。鼠标单击多维度视图卡片容器的多维度标签，打开原始感知数据视图，单击视图中光电维度的打开地图小视频按钮，并且单击跟踪按钮。完成视频监控跟踪功能 |

测试结果显示，系统能够正确进行设备显控以及联动，视频监控与跟踪功能。并且界面操作逻辑正确、流畅、无卡顿。测试结果如图 5-10,5-11 所示。

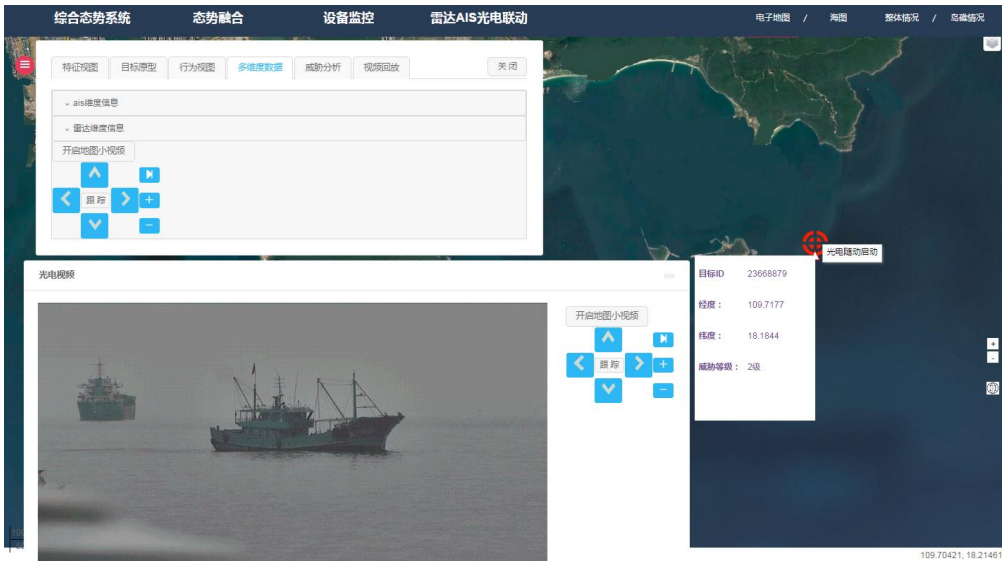

图 5-10 设备显控联动功能测试效果

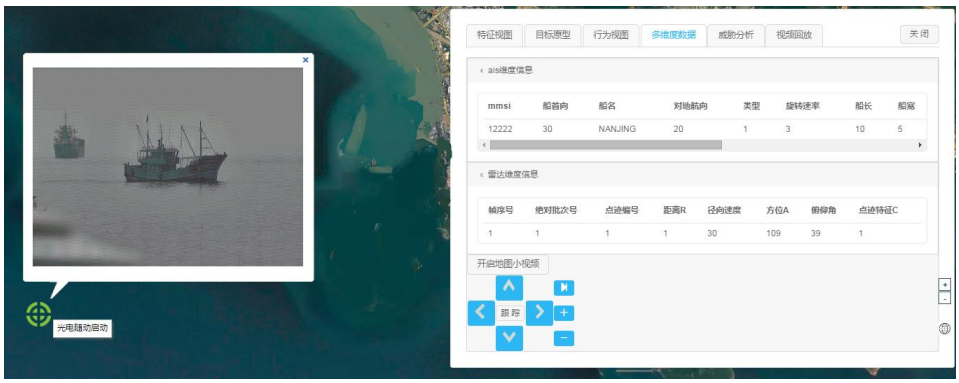


图 5-11 视频监控与跟踪功能测试效果

### 5.2.1.6 实时告警测试

对于实时监控模式实时告警功能显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-6 所示：

表 5-7 实时监控模式实时告警功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件，开启实时显控模式。<br>（3） 激活实时监控模式后，在地图上出现实时监控的目标标记点，当目标产生告警事件后，将会根据告警的等级不同进行不同程度的告警闪烁，并且根据告警等级的不同，标记变成红色或者黄色。 |

测试结果显示，系统能够实时的正确进行告警信息的显示，并且进行可靠的实时渲染闪烁效果，实时告警闪烁效果流畅无卡顿，并且界面操作逻辑正确、流畅、无卡顿。测试结果如图 5-12 所示。


图 5-12 实时监控模式实时告警功能测试效果

### 5.2.1.7 行为事件情景回放测试

对于实时监控模式行为事件情景回放功能显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-8 所示：

表 5-8 实时监控模式行为事件情景功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件，开启实时监控模式。<br>（5） 激活实时监控模式后，在地图上出现实时监控的目标标记点，单击目标标记，打开多维度数据视图卡片容器组件。<br>（3） 鼠标单击多维度视图卡片容器的行为视图标签，打开行为事件数据视图。<br>（4） 选择行为事件数据视图中的一个行为或者事件记录，单击行为事件情景回放按钮。 |

测试结果显示，系统能够实时的正确进行行为事件情景回放，行为事件情景回放效果流畅无卡顿，情景回放的信息完整，且控制回放的操作操作逻辑正确、流畅、无卡顿。测试结果如图 5-13，5-14，5-15 所示。

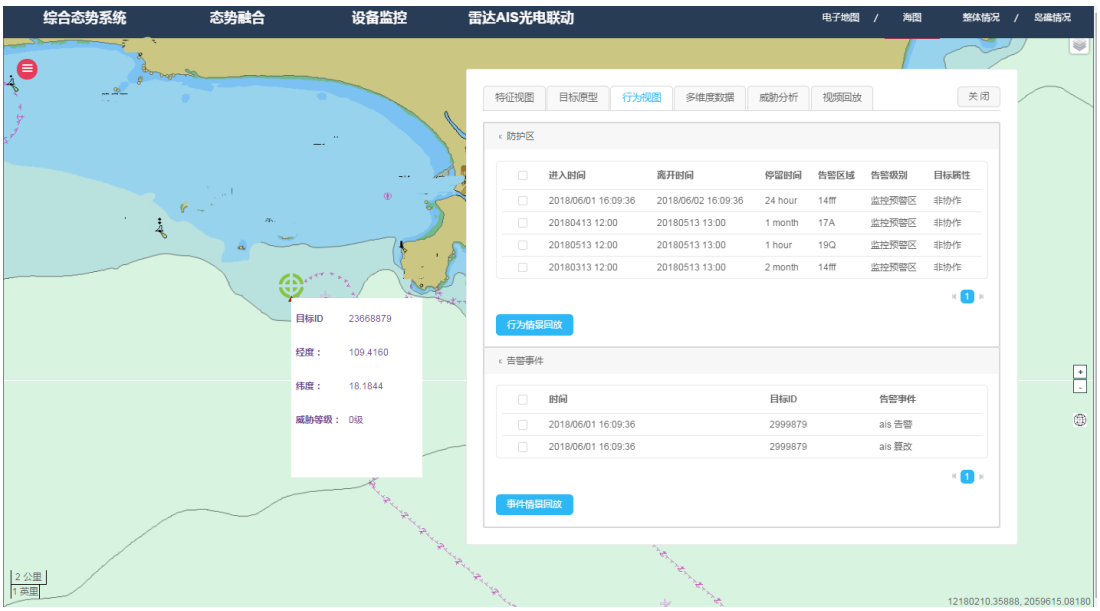


图 5-13 实时监控模式行为事件情景选择功能测试效果

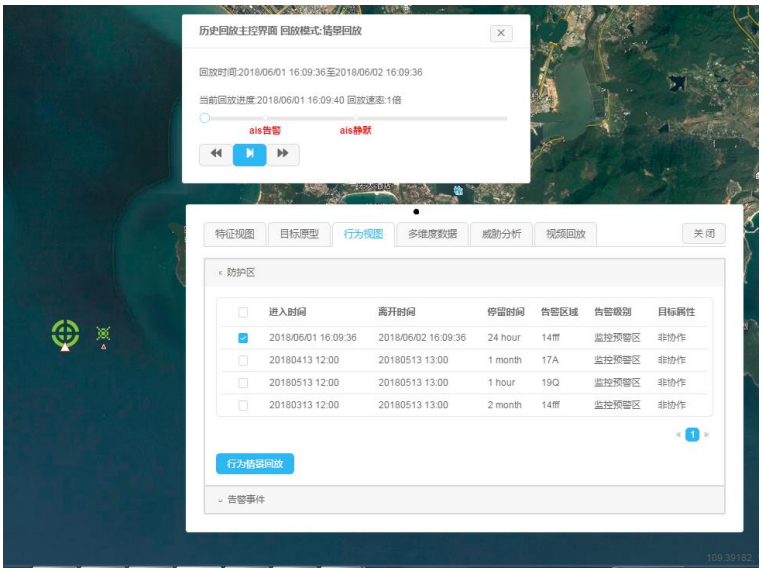


图 5-14 实时监控模式行为事件情景详细回放功能测试效果


图 5-15 实时监控模式行为事件情景回放进度控制组件

#### 5.2.1.8 视频情景回放测试

对于实时监控模式视频情景回放功能显示逻辑以及可视化效果进行功能测试，功能测试流程如表 5-9 所示：

表 5-9 实时监控模式视频情景功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击实时监控组件，开启实时监控模式。<br>（6） 激活实时监控模式后，在地图上出现实时监控的目标标记点，单击目标标记，打开多维度数据视图卡片容器组件。<br>（3） 鼠标单击多维度视图卡片容器的视频回放视图标签，打开视频回放视图。<br>（4） 选择视频回放列表的视频，单击开始播放按钮进行视频回放。 |

测试结果显示，系统能够正确进行视频情景回放，视频情景回放效果流畅无卡顿，情景回放的信息完整，且控制回放的操作操作逻辑正确、流畅、无卡顿。测试结果如图 5-16 所示。


图 5-16 实时监控模式视频情景回放测试效果

### 5.2.2 缓冲式批量数据加载算法测试

#### 5.2.2.1 批量数据缓冲测试

对于历史回放模式缓冲式批量数据加载算法进行测试，步骤如表 5-10 所示：

表 5-10 实时监控模式视频情景功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击历史回放中的目标回放组件，选择测试的回放目标与回放区间，开始历史回放。<br>（3） 开始进行历史回放后，在地图上出现历史回放的目标标记点，然后查看历史回放的时许与缓冲时间。 |

图 5-17,5-18 展示了在历史回放过程中，缓冲式批量数据加载算法的运行效果，一次缓冲 30 帧的感知数据，每一帧包含了 20 个目标的传感器综合信息。

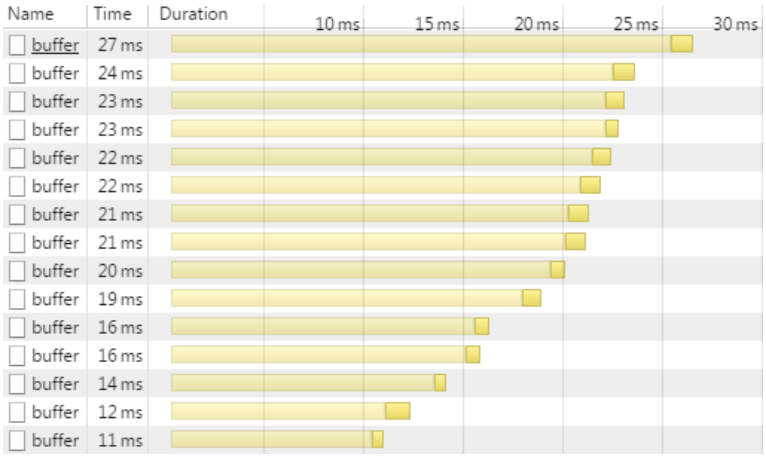


**图 5-17 缓冲式批量数据加载算法批量数据缓冲时间测试效果**

图 5-17 展示了进行 15 次缓冲的缓冲等待时间，每次缓冲 30 帧数据，平均缓冲时间为 19 毫秒。

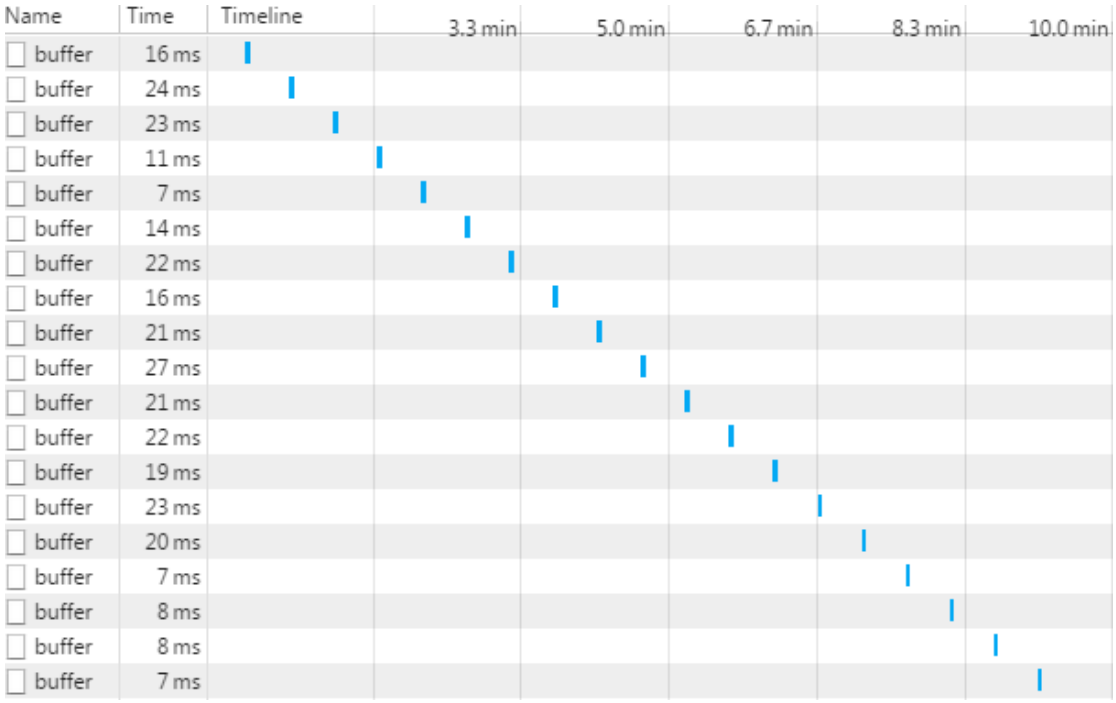


**图 5-18 缓冲式批量数据加载算法缓冲时序测试效果**

图 5-18 展示了 10 分钟内进行缓冲批量数据的时序图，其中相邻帧数据的时间差 1s,每次缓冲 30 帧数据，即根据缓冲批量数据算法得到上每 30 秒进行一次，实际效果为 10 分钟内缓冲了 19 次，符合算法预期。

测试结果显示对于更新频率在秒级别的历史回放模式中，使用缓冲批量数据加载算法缓冲的数据回传及时，不会产生明显卡顿，缓冲数据不会影响到用户体

验，达到了低时延要求。

#### 5.2.2.2 历史数据回放方式测试

对于历史回放模式历史数据回放方式进行测试，功能测试流程如表5-11所示：

表 5-11 历史回放模式历史数据回放方式功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击下拉组件列表历史回放组件中的相应方式的内部组件。<br>（3） 开始进行历史回放后，在地图上出现历史回放的目标标记点，查看该历史回放方式的效果。 |

测试结果显示，系统能够正确进行相应方式的历史数据回放，历史数据回放方式组件交互效果流畅无卡顿，回放的信息完整，且控制历史数据回放方式操作逻辑正确、流畅、无卡顿。测试结果如下。


图 5-19 四种历史数据回放方式测试效果

图 5-20 全景历史数据回放方式测试效果


图 5-21 目标历史数据回放方式测试效果

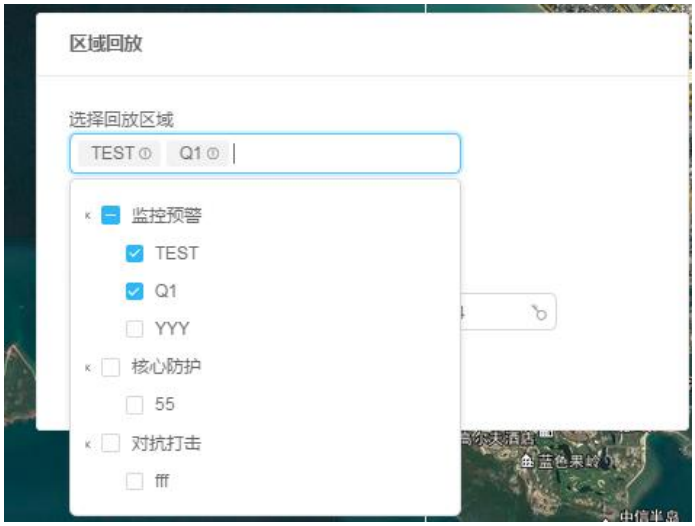


图 5-22 区域历史数据回放方式区域选择测试效果


图 5-23 区域历史数据回放方式区域绘制测试效果


图 5-24 区域历史数据回放方式区域回放测试效果


图 5-25 事件历史数据回放方式测试效果

### 5.2.2.3 历史回放流程控制测试

对于历史回放模式历史回放流程控制进行测试，功能测试流程如表5-12所示：

表 5-12 历史回放模式历史回放流程控制功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击下拉组件列表历史回放组件中的相应方式 |

的内部组件。

（3） 开始进行历史回放后，在地图上出现历史回放的目标标记点，查看该历史回放的效果。并且进行拖动进度条，暂停回放，加速减速等操作。

测试结果显示，系统能够正确进行相应方式的历史回放流程控制，历史回放流程控制交互效果流畅无卡顿，回放的信息完整，且历史回放流程控制操作逻辑正确、流畅、无卡顿。测试结果如下。


图 5-26 历史回放加速减速功能测试效果

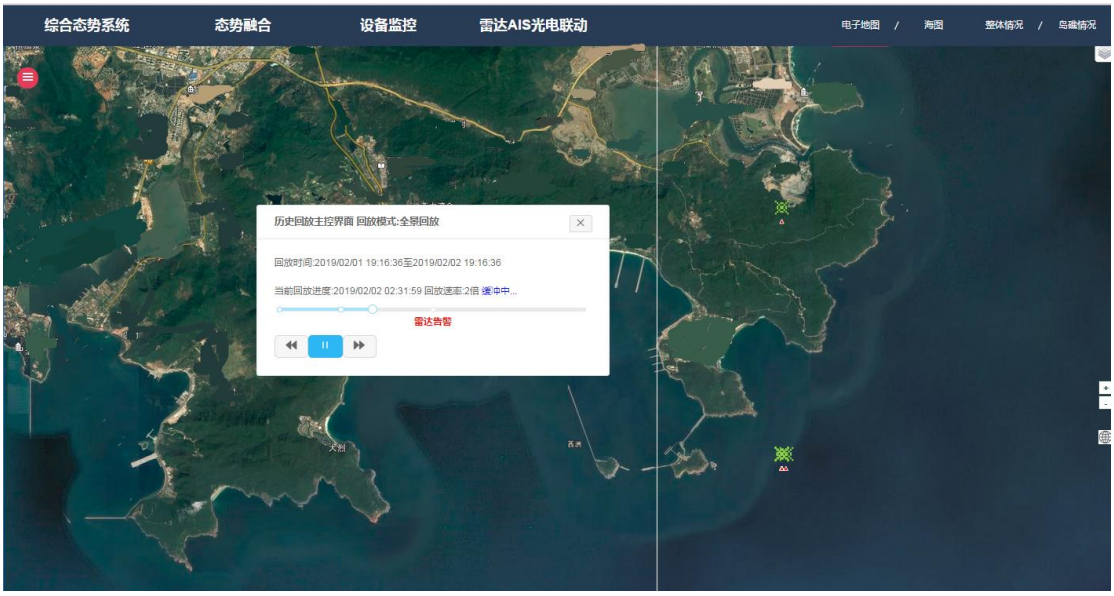


图 5-27 历史回放拖动进度条控制功能测试效果

### 5.2.2.4 事件告警测试

对于历史回放模式历史回放事件告警进行测试，功能测试流程如表 5-13 所示：

表 5-13 历史回放模式历史回放事件告警功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击下拉组件列表历史回放组件中的相应方式的内部组件。<br>（3） 开始进行历史回放后，在地图上出现历史回放的目标标记点，查看该历史回放的效果。并且进行加速操作。<br>（4） 等待历史回放至关键帧事件附近时候弹出事件告警通知。 |

测试结果显示，系统能够正确进行历史事件告警通知，历史事件告警通知交互效果流畅无卡顿，回放的信息完整，且历史事件告警通知操作逻辑正确。测试结果如下。


图 5-28 历史回放事件告警功能测试效果

### 5.2.3 跟踪判读服务及目标画像测试

#### 5.2.3.1 迁徙转移态势测试

对于跟踪判读服务迁徙转移态势进行测试，功能测试流程如表 5-14 所示：

**表 5-14 跟踪判读服务迁徙转移态势功能测试**

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击跟踪判读下拉组件中的迁徙态势按钮。<br>（3） 弹出对话框，选择进行空间数据查询的目标以及时间区间。 |

测试结果显示，系统能够正确进行跟踪判读服务迁徙转移态势渲染，跟踪判读服务迁徙转移态势交互效果流畅无卡顿，信息完整，且跟踪判读服务迁徙转移态势操作逻辑正确、流畅、无卡顿。测试结果如下。

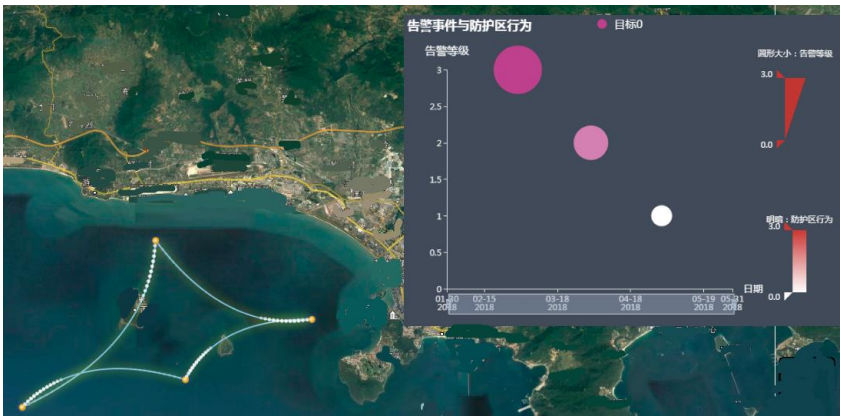


**图 5-29 迁徙转移态势功能测试效果**

#### 5.2.3.2 热点流动区域测试

对于跟踪判读服务热点区域进行测试，功能测试流程如表 5-15 所示：

**表 5-15 跟踪判读服务热点区域功能测试**

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击跟踪判读下拉组件中的热点区域按钮。<br>（3） 选择需要进行热点区域分析的目标以及时间区间。 |

测试结果显示，系统能够正确进行跟踪判读服务热点区域渲染，跟踪判读服务热点区域交互效果流畅无卡顿，信息完整，且跟踪判读服务热点区域操作逻

辑正确、流畅、无卡顿。测试结果如下所示。


图 5-30 热点区域功能测试效果

### 5.2.3.3 告警数据蜂窝测试

对于跟踪判读服务数据蜂窝进行测试，功能测试流程如表 5-16 所示：

表 5-16 跟踪判读服务数据蜂窝功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，单击跟踪判读下拉组件中的数据蜂窝按钮。<br>（3） 选择需要进行数据蜂窝分析的时间区间。 |

测试结果显示，系统能够正确进行跟踪判读服务数据蜂窝渲染，跟踪判读服务数据蜂窝交互效果流畅无卡顿，信息完整，且跟踪判读服务数据蜂窝操作逻辑正确、流畅、无卡顿。测试结果如图下所示。

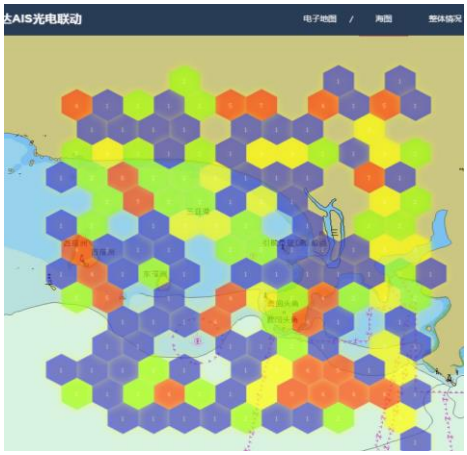

图 5-31 数据蜂窝功能测试效果

#### 5.2.3.4 目标画像功能测试

对于目标画像功能进行测试，功能测试流程如表 5-17 所示：

表 5-17 目标画像功能测试

| | |
|---|---|
| **功能测试流程** | （1） 开启实时与历史结合的多视图态势系统服务。<br>（2） 鼠标指针滑动至界面控制区态势融合标签，自动弹出下拉组件列表，进入实时监控模式并且打开多维度数据视图。<br>（3） 选择多维度数据视图的目标原型标签 |

测试结果显示，系统能够正确进行目标画像，目标画像交互效果流畅无卡顿，信息完整，且目标画像逻辑正确、流畅、无卡顿。测试结果如图 5-29 所示。

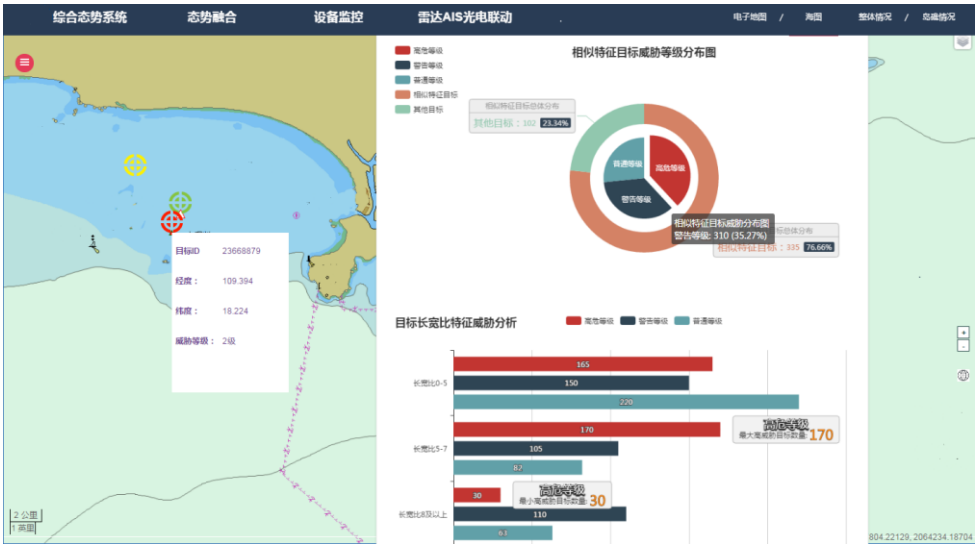


图 5-32 目标画像功能测试效果

## 5.3 系统性能测试

### 5.3.1 系统响应性能测试

当 Browser 端向后台请求首页的 HTML 文件、CSS 样式资源与 Javascript 脚本时候，从请求开始发出的时刻开始计时，一直到首页的资源文件完全加载完成的时刻，计算这两个时刻之间的差值即为系统首次首次响应时间。

我们在 Browser 刷新首页，直到首页完全加载后，再次刷新首页，这样一直持续十次，求十次首页刷新的平均系统首次响应时间

**表 5-18 刷新十次的系统平均首次响应时间**

| | |
|---|---|
| 平均系统响应时间 | 2.85s |
| 平均系统加载时间 | 19.1ms |
| 平均系统脚本运行时间 | 711ms |
| 平均系统渲染时间 | 73.16ms |
| 平均系统绘制时间 | 85.32ms |

其中一次测试的结果截图如下所示。

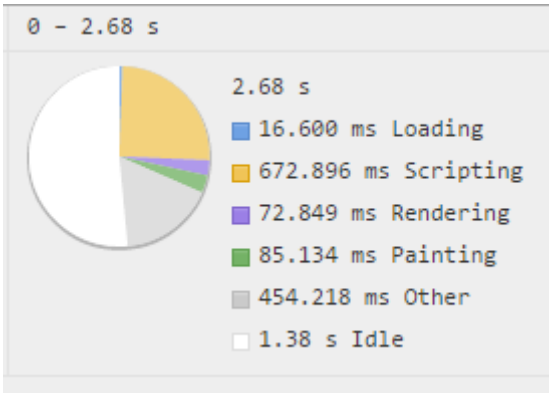


**图 5-33 系统响应时间饼状图**

对于系统的平均首次响应时间，存在一个时间判定原则，系统平均首次响应时间在 0-2 秒区间内，系统被评价称为优秀响应性能的系统，系统平均首次响应时间在 2-5 秒内，系统被评估称为良好响应性能的系统，系统平均首次响应在 5-10 秒内，则其被评估为及格性能系统。依据这个时间判定原则，本系统被评估为良好响应性能系统

### 5.3.2 系统稳定性测试

态势呈现系统需要具备长期运行稳定性，以便于对区域态势进行长期监控。

首先开启系统的实时与历史服务，保持系统服务执行的状态，等待一个月后，对于系统首次响应时间进行循环重复记录十次后，求平均值得到如下表格：

**表 5-19 运行一个月后十次系统首次响应平均时间**

| | |
|---|---|
| 平均系统响应时间 | 4.0s |

| | |
|---|---|
| 平均系统加载时间 | 30ms |
| 平均系统脚本运行时间 | 1.5s |
| 平均系统渲染时间 | 80ms |
| 平均系统绘制时间 | 100ms |

其中一次测试的结果截图如图 5-34 所示。

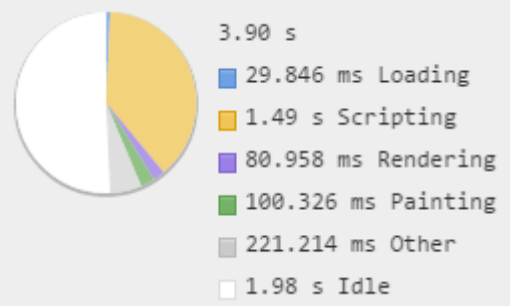


**图 5-34 运行一个月后其中一次首次响应图**

实验结果表明，系统运行了一个月后的平均响应时间中可以看出浏览器渲染效率性能稍微降低，这直接造成了加载时间的延长，但是总体上仍然表现良好，测试结果证明，系统响应时间具备稳定性。

系统运行一个月后记录下来服务器的 CPU、磁盘、内存占用。

**表 5-20 系统运行一个月后的服务器 CPU、磁盘、内存**

| 内存 | CPU | 磁盘 |
|---|---|---|
| 577MB（1.76%） | 20% | 767KB/s |

CPU 占用率如图下所示.

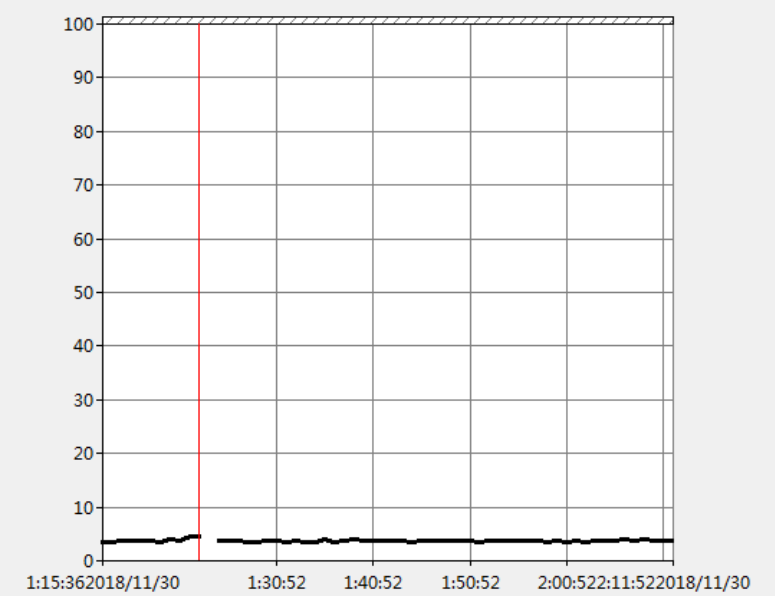


图 5-35 系统运行一个月后运行后 CPU 占用率

从表 5-35 中结果可知，系统的磁盘 I/O 速率与内存占用、CPU 占用都在正常的范围内，系统运行正常，并没有损坏磁盘和出现堆耗尽与内存泄露的情况。系统稳定运行。

如图 5-36 所示。使用 Browser 端的渲染性能测试工具，对堆空间占用情况进行测试，测试图为周期性的直角三角形，说明了在渲染的过程中，系统标记节点集合会周期性的进行图层清理，通过清理上一次图层渲染的过时效性的数据，保证堆空间的最大程度利用，并且防止出现堆栈耗尽、内存泄露。

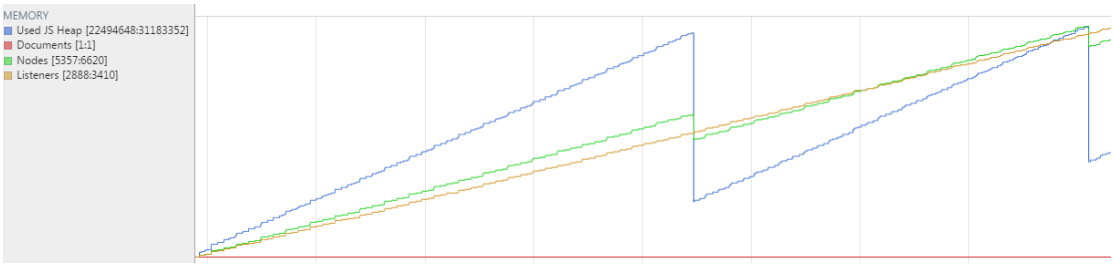


图 5-36 javascript 堆占用图

### 5.3.3 系统并发缓冲性能测试

我们使用 Apache JMeter 压力测试工具模拟巨大的负载，不断的对于系统缓冲批量数据的接口进行并发请求，依次测试缓冲批量数据的时间窗口、并发量、系统吞吐量之间关系。

首先系统分别使用 50 个线程同时对后台进行访问，轮询间隔 20 次，100 个线程同时对后台访问轮询 10 次，1000 个线程对后台进行访问，轮询 1 次的测试

方法，画出系统的平均响应时间、方差、吞吐量随时间的变化规律。其中缓冲数据时间窗口为 150，进行并发的性能测试，测试结果如下图所示。

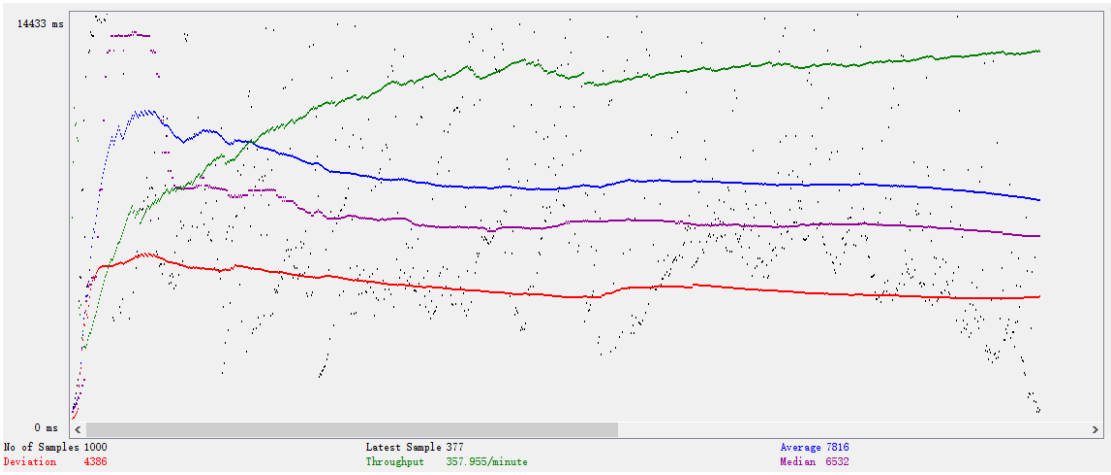


图 5-37 50 并发后台性能图

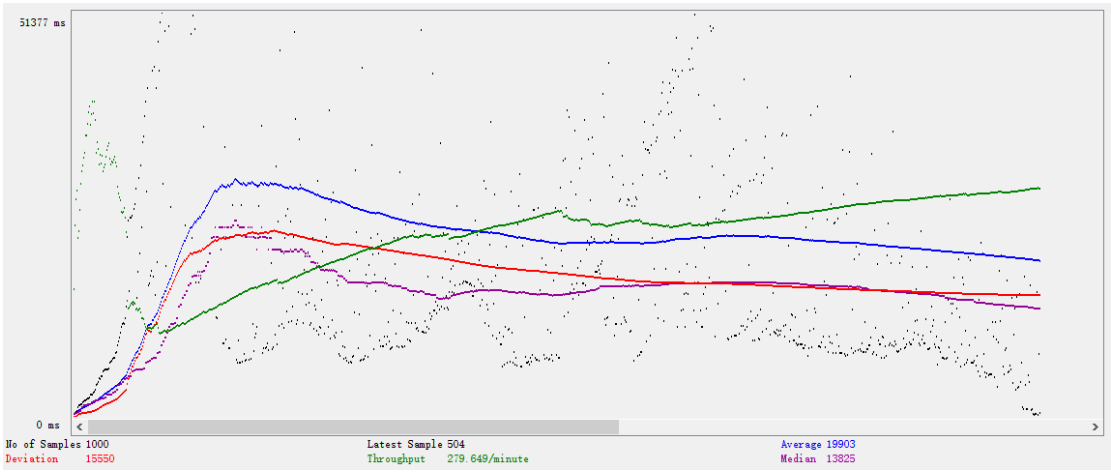


图 5-38 100 并发后台性能

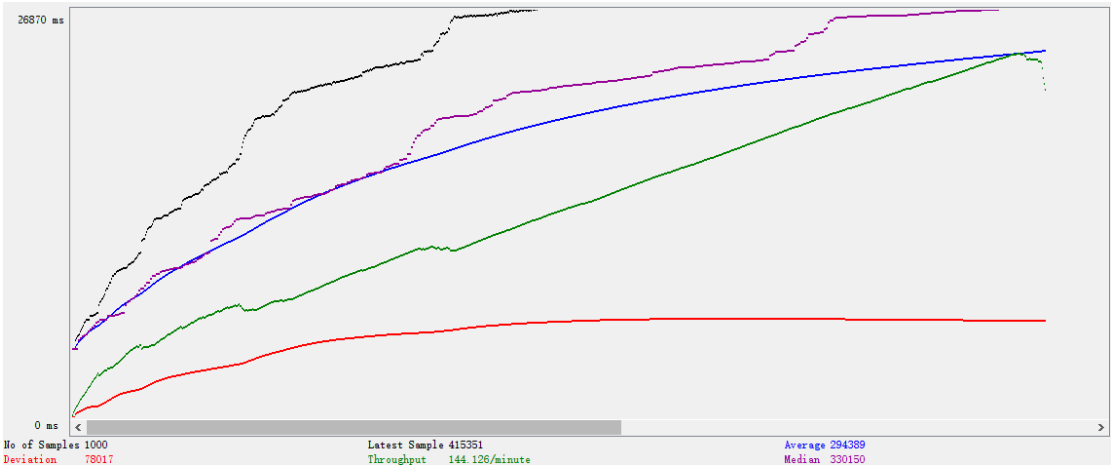


图 5-39 1000 并发后台性能

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Maximum | Error % | Throughput | Received K... | Sent KB/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request | 2000 | 14655 | 12948 | 25268 | 25390 | 33467 | 288 | 42156 | 26.20% | 1.2/sec | 151.33 | 0.22 |
| TOTAL | 2000 | 14655 | 12948 | 25268 | 25390 | 33467 | 288 | 42156 | 26.20% | 1.2/sec | 151.33 | 0.22 |

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Maximum | Error % | Throughput | Received K... | Sent KB/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request | 1000 | 19903 | 13825 | 44015 | 57175 | 66105 | 248 | 84377 | 0.00% | 4.7/sec | 1127.36 | 1.33 |
| TOTAL | 1000 | 19903 | 13825 | 44015 | 57175 | 66105 | 248 | 84377 | 0.00% | 4.7/sec | 1127.36 | 1.33 |

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Maximum | Error % | Throughput | Received K... | Sent KB/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request | 2000 | 157146 | 67084 | 351496 | 358976 | 368961 | 248 | 415351 | 0.00% | 2.1/sec | 501.31 | 0.59 |
| TOTAL | 2000 | 157146 | 67084 | 351496 | 358976 | 368961 | 248 | 415351 | 0.00% | 2.1/sec | 501.31 | 0.59 |

图 5-40 聚合分析

由上面的图所示，系统的随着时间的流动，方差趋于稳定，说明了系统逐渐达到一个稳态，在 50 并发与 100 并发的实验中，吞吐量先上升后下降说明了，说明了系统符合了后台资源分配模型。平均响应时间先上升后趋于稳定。

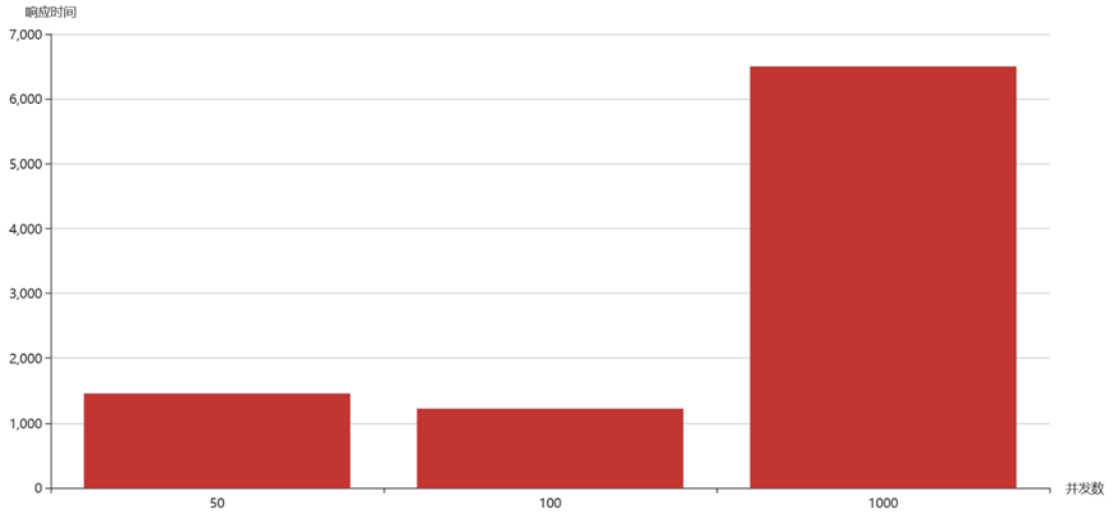


图 5-41 不同并发数后台的响应时间对比

可以看到，当并发数增多后，后台的响应时间有了下降，说明后台使用了合理的负载均衡分配方案，提高了效率，而并发数过多以后，超过了后台支撑的阈值，因此响应时间变为很大。

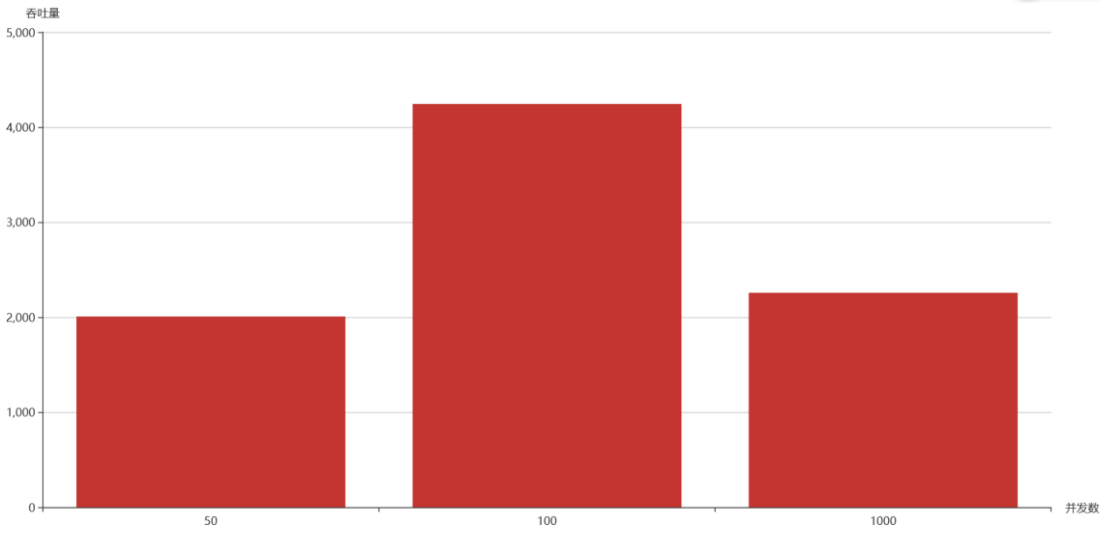


**图 5-42 不同并发数后台的吞吐量对比**

可以看到，当并发数增大多后系统的吞吐量先增加后减少，说明了系统使用了资源适配模型，合理调度配置资源，但是超过系统的阈值之后，就会下降。

分别设置缓冲时间窗口为 10 与 150，重新做上述的不同并发下，吞吐量、响应时间的值变化实验。实验结果如下。

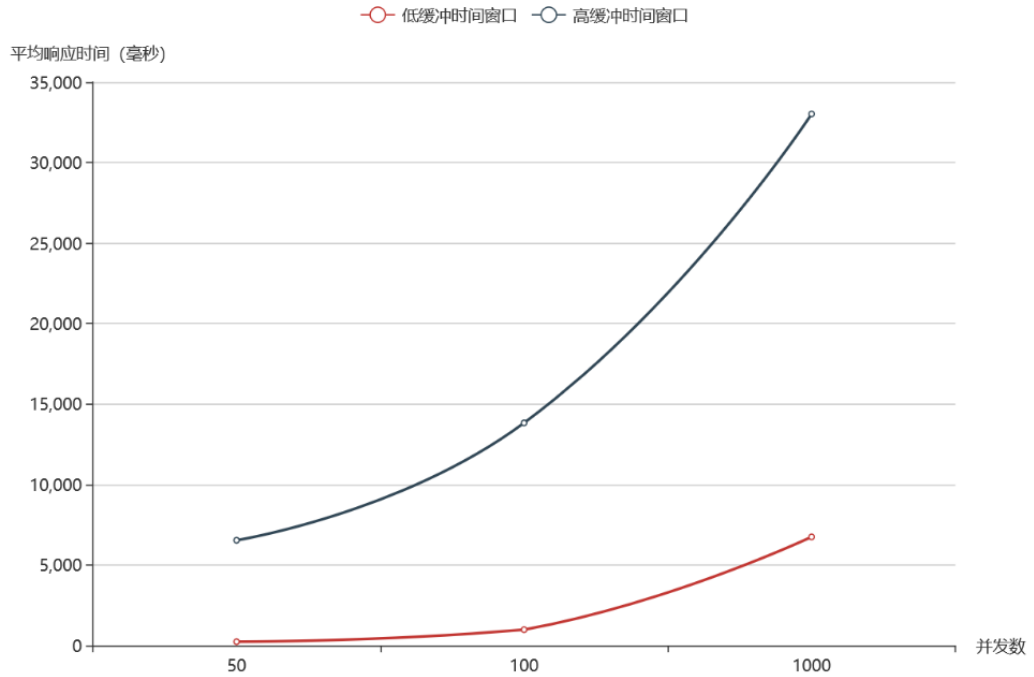

图 5-43 不同缓冲窗口不同并发的响应时间对比

图显示了系统在 50，100，150 增加并发的过程中平均响应时间曲线相对于超参数缓冲时间窗口的对比，此时低缓冲时间窗口大小为 10，高缓冲时间窗口大小为 150，说明当缓冲时间窗口降低时，将会提高系统并发的响应时间，说明当缓冲时间窗口提高时，将会降低系统并发的性能。

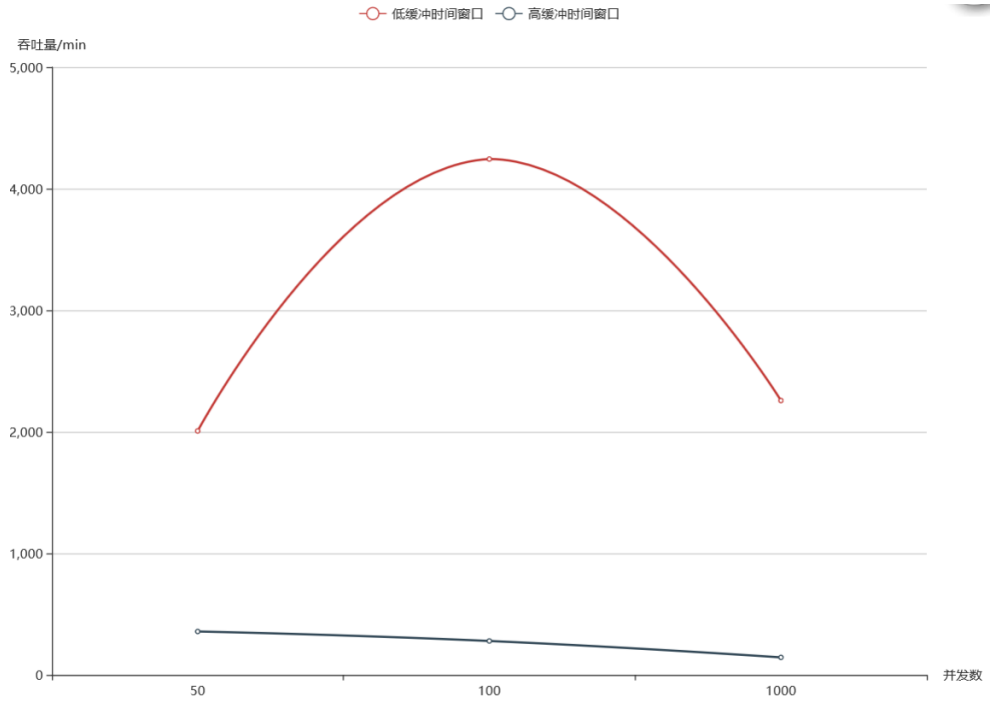


图 5-44 不同缓冲时间窗口不同并发的吞吐量时间对比

图显示了系统随着并发度的增加，在不同缓冲时间窗口，吞吐量的变化趋势，此时低缓冲时间窗口大小为 10，高缓冲时间窗口大小为 150。说明在低缓冲时间窗口时，并发数对于系统的吞吐量稳态有较大影响，随着并发数目的增加，吞吐量先升高再降低，并且，缓冲时间窗口也会影响吞吐量，缓冲时间窗口越小，相对吞吐量越大。

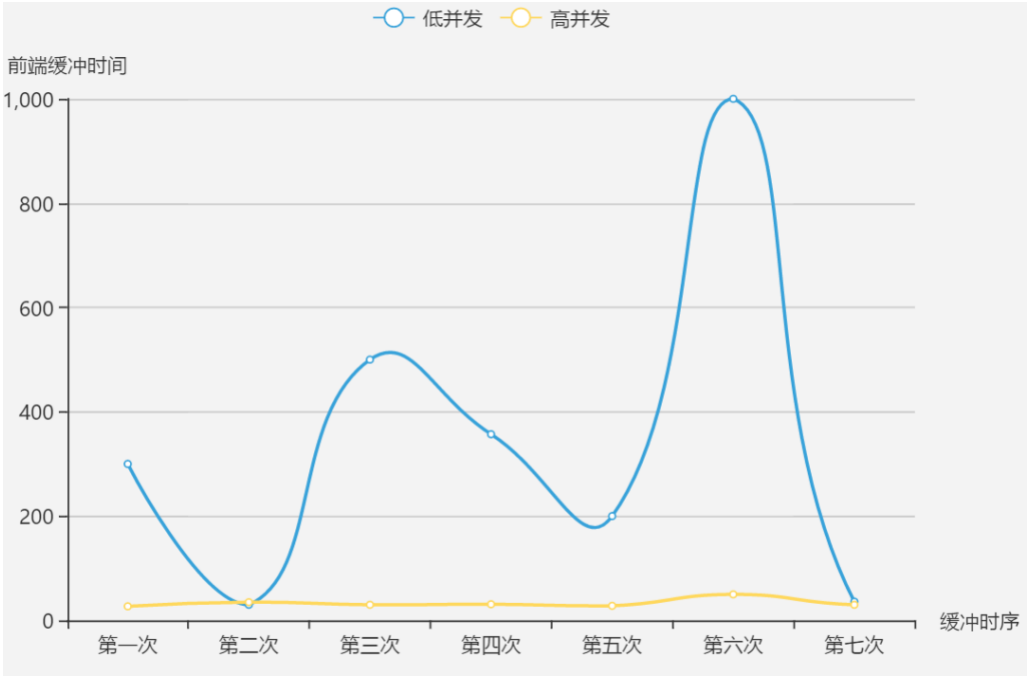


**图 5-45 不同并发的缓冲时间对比**

图显示了随着缓冲时间的增加，并发数与前端缓冲时间的影响，图说明了，并发数目越高，前端缓冲时间越不稳定，时间越长，并发数越低，并发数目越低，前端的播放越流畅。

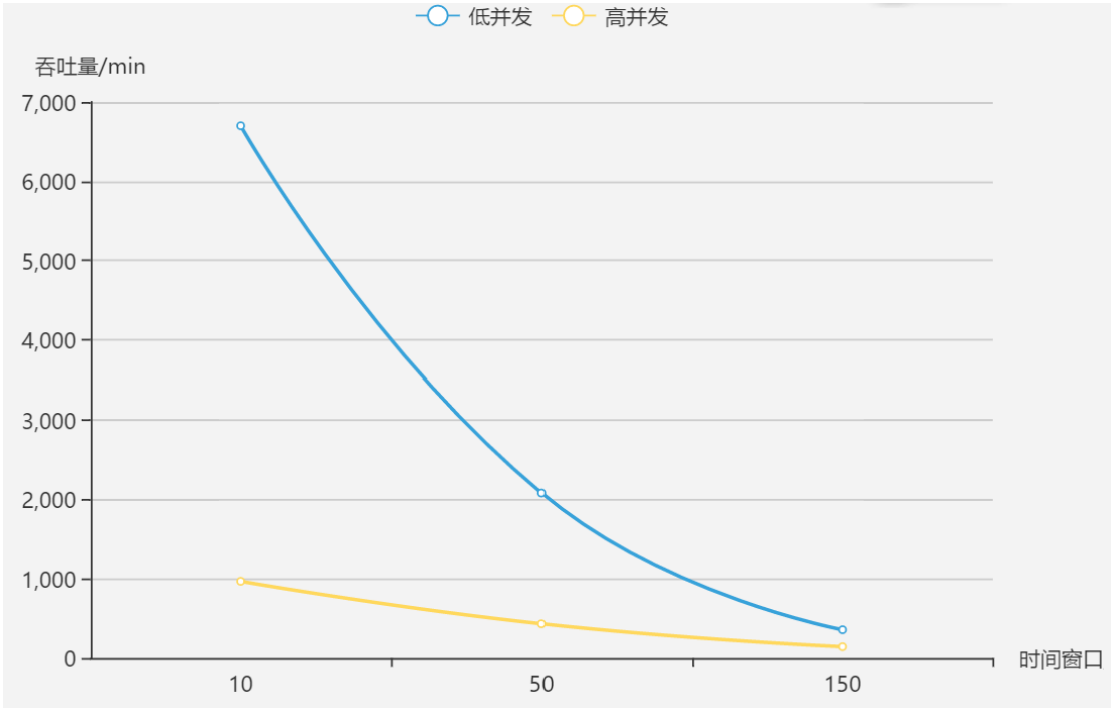


**图 5-46 不同时间窗口的吞吐量对比**

图显示了随着缓冲时间窗口的增加，并发数对吞吐量的影响，此时低缓冲时间窗口大小为 10，高缓冲时间窗口大小为 150，可以看到缓冲时间窗口越大，吞

吐量越小，而并发数目越高，相同的缓冲时间窗口内，吞吐量差异越小，并发数据越低，在相同的缓冲时间内吞吐量差异越大。

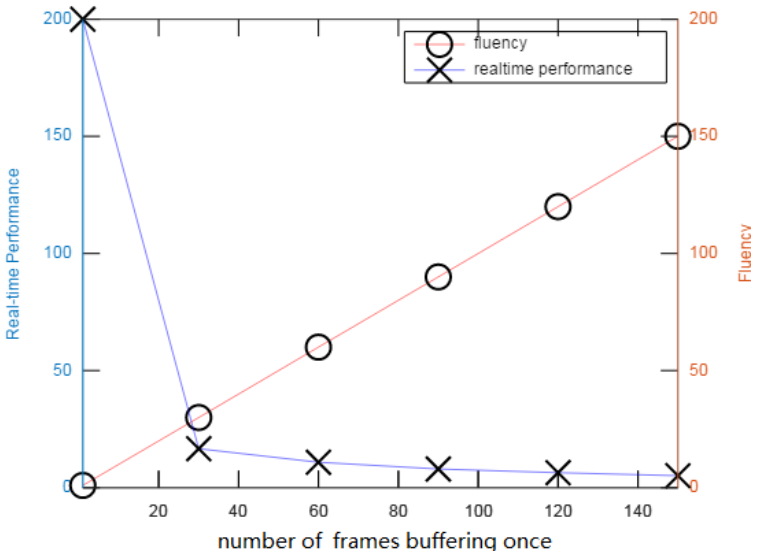


**图 5-47 缓冲时间窗口参数调优图**

在此次实验中，不同的缓冲时间窗口参数会对历史回放的实时性和流畅性产生影响。图 5-47 展示了以不同缓冲时间窗口参数进行缓冲批量数据的相对实时性与流畅性。

非常明显，当缓冲时间窗口参数增加时，流畅性会上升，但是实时性会下降，因为缓冲数据的数量增加导致缓冲时延增大，从而降低了回放实时性。

测试结果表明，需要一个合适的缓冲时间窗口，既能保持一定的回放实时性，又可以保证流畅性。本系统中设置缓存时间窗口为 30。即两条曲线的交汇点。图 5-47 表明了当前设置缓存时间窗口为 30 的时候平均缓冲时间为 20ms，具备较好的实时性，同时也保证了流畅性。

## 5.4 本章小结

本章通过对系统功能和性能进行测试，来达到验证系统有效性、可用性的目标。先对系统的所有功能进行测试，然后从系统响应性能、稳定性能、并发缓冲性能角度分析了系统的性能稳定性，最终验证统的功能完备、性能稳定，符合实时历史双展示模式的态势呈现要求。

# 第六章 总结与展望

## 6.1 工作总结

本课题研究实现了一个基于目标信息图谱、实时与历史结合的多视图物联网态势显控系统，系统会将物联网与云 GIS 技术相结合，以目标为核心对多种监测数据进行关联、融合、挖掘分析，从而形成目标信息图谱与多角度数据视图，最终基于时空范式与实时与历史结合的大数据可视化技术实现一个性能优良、美观、人机交互友好的可视化界面。

首先对于系统进行总体设计，确定了态势呈现系统的整体架构设计。并且设计了系统的目标是实现基于目标信息图谱和多数据视图的目标实时监控与历史回放与基于 GIS 服务的目标跟踪判读及目标画像。通过时空范式形成一种实时历史双展示模式的态势呈现系统。设计部分总结了系统的总体服务结构，从服务交互的角度解释实时与历史结合的思想，总结了实时服务与历史服务的相辅相成的关系。通过实时服务与历史服务结合的角度设计了总体服务结构与总体功能结构。为了实现系统服务解耦，本文采用了基于 React 框架的 MVVM 组件化架构，对于系统进行了总体组件化设计。最终梳理了系统服务、组件、功能的上下行数据流与整体流程，完成系统的总体设计工作。

总体设计思想确定为实时与历史结合后，本文主要通过时空范式对态势呈现系统进行详细的组件化实现。基于时间维度的细粒度、局部数据可视化实现方法，形成了实时历史双展示模式的组件化方案与具体实现算法。基于空间范式的粗粒度、全局性的空间位置大数据可视化方法实现了跟踪判读与目标画像渲染服务中的渲染算法与组件化架构详细实现方案。

最终经过逐个功能，逐个步骤的系统功能测试，与系统的全面性能分析测试后，证明态势呈现系统可以解决当下态势监控的痛点，实现态势信息展示的关联性，对目标的认知具备一定深度，并且可以从全局粗粒度的角度认知目标的历史空间属性，最终成为一个具备了实时历史双展示模式的态势呈现系统。

## 6.2 工作展望

态势系统已经具备了稳定性与性能可靠性，但是仍然有潜力继续演化成为更高维度、更具备智能化的态势认知系统。实现系统的进化还需要做的工作如下：

1）态势呈现系统中的目标画像仅仅是初步的进行了目标信息标签化，且只

针对了其中两个标签做数据聚合形成数据立方，未来的智能态势认知系统，应该根据可以根据需要进行任意标签的聚合数据立方，并且通过智能的自规划画像算法，将目标信息库中的目标划分为具备为相同共性的多个目标凝集子群，形成目标的群体画像，群体感知。

2）本系统的时间维度下的历史轨迹数据可以为实时态势监控提供结合了历史挖掘信息的情报支持，但不提供空间维度的历史粗粒度位置数据对于实时态势监控的情报支持，未来可以开展基于时空范式结合的历史挖掘数据支撑实时监控方案。

3）本系统仅仅支持基于目标信息库的历史回放，未来实现了对目标的群体感知后，需要支持共性目标凝集子群的群体历史行为、群体历史事件、群体区域事件回放，即实现基于目标群态势认知的群体意识历史回放。

# 参考文献


[1] 张明杰,王静平.态势实时显示与回放系统的设计与实现[J].指挥控制与仿真,2019(01):112-119.

[2] 高翔,陈贵凤,赵宏雷.基于数据挖掘的电力信息系统网络安全态势评估[J/OL].电测与仪表:1-6[2019-02-27].

[3] 孙峥皓,杨利民,廖馨,张琦.关于虚拟战场态势仿真系统设计研究[J].计算机仿真,2018,35(12):309-312+324.

[4] 雷进宇,初秀民,蒋仲廉,王乐.基于可视分析的船舶航行态势感知系统设计[J].中国航海,2018,41(03):47-52.

[5] Frédérique Mayer. Exploring the notion of situation for responsive manufacturing systems specification issues[J]. IFAC PapersOnLine,2018,51(11).

[6] 王晓芳. 我国工业控制系统威胁发现与态势感知[A]. 天津市电子学会、天津市仪器仪表学会.第三十二届中国（天津）2018’IT、网络、信息技术、电子、仪器仪表创新学术会议论文集[C].天津市电子学会、天津市仪器仪表学会:天津市电子学会,2018:4.

[7] 张明杰.一种基于 B~+树航迹态势回放系统的设计与实验[J].计算机应用与软件,2018,35(08):159-164.

[8] 潘辰. 基于 WebGIS 的轻量级态势显示系统设计与实现[D].北京邮电大学,2018.

[9] 廖月.基于知识发现的网络安全态势感知系统[J].电脑迷,2018(03):71-72.

[10] 张昊.通用战场态势可视化系统的设计及实现[J].计算机工程与应用,2018,54(17):258-265.

[11] B. Cheng, D. Zhu, S. Zhao, and J. Chen, Situation-aware iot service coordination using the event-driven soa paradigm, IEEE Transactions on Network and Service Management, vol. 13, no. 2, pp. 349361, June 2.

[12] G. W. Liu, Y. X. Liu, Y. G. Ji, and C. Wang, Track association for highfrequency surface wave radar and AIS based on fuzzy double threshold theory, Systems Engineering and Electronics, vol.38, no. 3, pp. 558.

[13] Y. L. Zhang, and J. Y. Wang, Overview of the Multi-sensor Data Fusion Technology, Ship Electronic Engineering, vol.33, no. 2, p. 41, 2013.

[14] J. Kang, The key technology research of Multi-sensors Information Fusion, Harbin Engineering University, 2013.

[15] J. Dong, Research and implementation on support batch computing and str eaming computing Big Data System, Northwest University, 2015.

[16] X. Y. Zhu, and P. P. Pang, Review of Big Data Stream Computing Syste m, Group Technology & Production Modernization, no.4, pp. 5053, 2016.

[17] Duan Y , Shao L , Hu G . Specifying Knowledge Graph with Data Grap h, Information Graph, Knowledge Graph, and Wisdom Graph[C]// IEEE Inte rnational Conference on Software Engineering Research. IEEE, 2017.

[18] Information on https://en.wikipedia.org/wiki/Model-view-controller

[19] Zhang D , Wei Z , Yang Y . Research on Lightweight MVC Framework Based on Spring MVC and Mybatis[C]// Sixth International Symposium on Computational Intelligence and Design. IEEE, 2014.

[20] Li X , Chang D L , Peng H , et al. Application of MVVM design patter n in MES[J]. 2015.

[21] Information on https://github.com/facebook/react/

[22] Jia PingLiu Juhai,Wang Yuan. GIS Based on Cloud Computing and Intern et of Things [J]. Information Forum,2012,6

[23] Information on https://ant.design/docs/react/introduce-cn

[24] Su Y Y , Liu P C , Kao C C . Efficient Buffer Output Control for Mult imedia Stream Playback[C]// IEEE International Symposium on Multimedia. IEEE Computer Society, 2013.

[25] Shuai Bai, Zhiqun He, Tingbing Xu, Zheng Zhu, Yuan Dong, Hongliang Bai Multi-hierarchical Independent Correlation Filters for Visual Tracking [D B/OL]. https://arxiv.org/abs/1811.10302

# 作者攻读学位期间发表的学术论文

[1] Design and Implementation of Real Time and History multi-view IoT trend Display and Control System[C]//IEEE 8th Joint International Information Technology and Artificial Intelligence Conference.2019.