TrackPoint-青训营大项目项目提交文档

一、项目介绍

项目信息: **TrackPoint** —— 一个简易的跟踪用户在网站中行为的数据采集服务平台。本项目旨在开发一个完整的埋点研发体系，提供项目用户行为分析、性能监控、报警监控的能力。

项目仓库地址: https://github.com/wewb/nomad_sdk

二、项目分工

团队成员	主要贡献
吴佳晟	个人项目

三、项目实现

3.1 技术选型与场景分析

3.1.1 系统架构概述

该项目采用了前后端分离的三层架构设计:

1. 数据采集层(track-sdk)

- 基于TypeScript开发的前端SDK
- 支持自动和手动埋点
- 提供错误监控、用户行为跟踪等核心功能

2. 服务端层(track-server)

- 基于Node.js + Express + MongoDB的后端服务
- 提供数据接收、存储和分析能力
- RESTful API设计

3. 可视化平台层(track-platform)

- 基于React + TypeScript的管理平台

- 提供数据可视化、应用管理等功能
- 采用TDesign组件库构建UI

3.1.2 关键技术选型

1. 前端SDK(track-sdk):

- TypeScript: 提供类型安全和更好的开发体验
- 模块化设计: 支持按需引入
- 自动化数据采集
- 防抖/节流优化

2. 后端服务(track-server):

- Node.js + Express: 高性能异步I/O
- MongoDB: 适合存储非结构化的埋点数据
- JWT认证: 确保API安全性

3. 管理平台(track-platform):

- React + TypeScript: 现代化前端框架
- TDesign: 企业级UI组件库
- ECharts: 数据可视化

3.1.3 系统规模预估

1. 存储需求预估:

- 单条事件数据约500字节
- 假设日均PV 100万:
  - 日存储量: 500B * 100万 ≈ 500MB/天
  - 年存储量: 500MB * 365 ≈ 180GB/年
- 建议预留1TB存储空间(含索引、备份等)

2. 服务器配置建议:

- 应用服务器:

- 2核4G内存(小规模)
- 4核8G内存(中等规模)
- 8核16G内存(大规模)

- MongoDB服务器:
  - 4核8G内存起步
  - 根据数据量增长扩容

3. 并发处理能力:

- SDK端采用批量上报机制(默认20条/patch，这里可以根据服务需求修改SDK代码来增加或减少批次数量)
- 服务端预估处理能力:
  - 单机并发: 1000 QPS
  - 峰值并发: 2000 QPS
- 可通过负载均衡扩展

3.1.4 扩展性考虑

1. 水平扩展:

- 应用服务器可多实例部署
- MongoDB支持分片集群
- 可引入消息队列削峰

2. 数据处理:

- 支持实时分析和离线分析
- 预留大数据处理接口
- 数据分级存储机制

3. 功能扩展:

- 插件化架构设计
- 预留自定义分析模块
- 支持第三方数据导出

3.1.5 性能优化策略

1. 前端SDK:

- 数据批量上报
- 本地缓存机制
- 网络状态感知

2. 后端服务:

- 数据库索引优化
- 缓存层设计
- 异步处理机制

3. 管理平台:

- 按需加载
- 虚拟列表
- 数据缓存

这些技术选型和架构设计基于系统的实际需求和可扩展性考虑。随着业务规模的增长，可以通过扩展服务器、优化存储结构等方式进行扩容。

3.2 架构设计与场景分析

3.2.1 用户场景分析

1. 用户分层

- 普通用户 (99.5%):
  - 日均浏览量: 20-50 PV
  - 数据上报频率: 较低
  - 存储需求: 较小
- 活跃用户/大V (0.5%):
  - 日均浏览量: 200+ PV
  - 频繁数据上报
  - 大量自定义事件

    - 存储需求: 较大

2. 关键场景假设

- 高峰期访问: 工作日 10:00-11:30, 14:00-15:30
- 数据上报峰值: 预计为平均值的3倍
- 大V用户单日产生事件量约为普通用户的10倍

3.2.2 系统架构设计

1. 多层架构设计

```
Web application ---------------[用户层]
    ↓
track-sdk ---------------------[接入层]
    ↓
track-server (Node.js集群) ---[应用层]
    ↓          ↑
MongoDB集群          -----------[数据层]
    ↓ → 数据分析服务
```

2. 数据流设计

```
数据采集 → 预处理 → 实时处理 → 存储
   ↓          ↓          ↓          ↓
SDK采集    数据清洗    实时分析    分层存储
   ↓          ↓          ↓          ↓
批量上报    格式校验    聚合计算    冷热分离
```

3.2.3 关键问题解决方案

1. 大V用户高并发处理

- 实现方案:
    - 独立数据分片
    - 专用缓存层
    - 优先级队列处理
- 技术细节:

```
用户请求 → 身份识别 → 优先级判断
```

```
2       ↓           ↓           ↓
3    批量处理    缓存命中      队列调度
```

2. 数据存储优化

- 分层存储策略:
  - 热数据: Redis缓存
  - 温数据: MongoDB主库
  - 冷数据: MongoDB从库
- 数据生命周期:

```
1   实时数据(24h) → 近期数据(7天) → 历史数据(90天+)
2        ↓              ↓              ↓
3     Redis缓存     MongoDB主库     归档存储
```

3. 高可用设计

- 服务器集群:
  - 最少3台应用服务器
  - MongoDB 1主2从架构
- 故障转移:
  - 服务自动发现
  - 负载自动调整
  - 数据自动同步

3.2.4 扩展性设计

1. 垂直扩展

- 应用层:
  - 动态服务扩容
  - 多实例部署
- 数据层:
  - 分片集群
  - 读写分离

2. 水平扩展

```
                          → 应用服务1
负载均衡器 (LB) → 应用服务2
                          → 应用服务3
```

3.2.5 性能优化方案

批量处理机制

- SDK端:

```
事件收集 → 本地缓存 → 批量上报(20条/次)
   ↓          ↓           ↓
防抖处理   IndexedDB   网络优化
```

- 服务端:

```
请求接收 → 任务队列 → 批量处理 → 存储
   ↓          ↓          ↓         ↓
限流控制   优先级排序   聚合计算   异步写入
```

3.2.6 监控告警机制

1. 系统监控

- 服务器资源监控
- 接口响应时间
- 错误率监控

2. 业务监控

- 数据上报量
- 用户活跃度

3. 告警策略

- 多级告警机制
- 自动扩容触发
- 故障自动转移

这个架构设计充分考虑了不同用户群体的使用特点。通过分层架构和优先级处理机制，确保了系统在高并发情况下的稳定性和可扩展性。同时，采用多级缓存和数据分层存储策略，优化了系统性能和资源利用率。

3.3 项目代码介绍

@track-server 服务端

track-server 是项目的后端服务,基于 Node.js + Express + MongoDB 构建。主要包含以下核心模块:

1. 数据模型(models):

- User.js - 用户模型,包含用户认证和权限管理
- Project.js - 应用项目模型,管理项目基本信息和端点配置
- Event.js - 事件模型,存储采集的用户行为数据
- Session.js - 会话模型,记录用户访问会话
- ApiKey.js - API密钥模型,用于SDK接入认证

2. 路由处理(routes):

- auth.js - 处理用户登录注册等认证相关请求
- application.js - 处理应用管理相关请求
- track.js - 处理事件数据采集和查询请求
- statistics.js - 处理数据统计分析相关请求
- api.js - 处理SDK接入的API请求

3. 中间件(middleware):

- auth.js - 用户认证中间件
- apiAuth.js - API认证中间件
- validate.js - 请求参数验证中间件
- projectAccess.js - 项目访问权限控制中间件

4. 服务(services):

- StatisticsService.js - 提供数据统计分析能力,包括:
  - 用户行为分析

- 时间维度分析
- 漏斗分析
- 路径分析
- 事件分析等

@track-sdk 客户端SDK

track-sdk 是一个轻量级的前端埋点SDK,提供了简单易用的数据采集接口。主要特性:

1. 核心功能(track-point.js):

```
class TrackPoint {
  // 初始化配置
  register(config: TrackConfig)

  // 发送事件数据
  sendEvent(eventName: EventName, params: EventParams)

  // 自动采集
  setupActionTracking()

  // 环境信息采集
  collectUserEnvInfo()

  // 数据上报
  flushEvents()
}
```

2. 支持的事件类型(types.ts):

```
enum EventName {
  // 内置组件
  CLICK_EVENT,       // 点击事件
  PAGE_VIEW_EVENT,   // 页面访问
  PAGE_LEAVE_EVENT, // 页面离开
  ERROR_EVENT,       // 错误事件
  SEARCH_EVENT,      // 搜索事件
  SHARE_EVENT       // 分享事件
  // 以及自定义事件
}
```

3. 主要特性:

- 支持自动采集和手动埋点
- 支持自定义公共参数
- 支持错误监控
- 支持会话管理

@track-platform 管理平台

track-platform 是一个基于 React + TypeScript 开发的数据管理平台。主要模块:

1. 应用管理(/pages/Applications):

- 应用列表展示
- 新建应用
- 应用详情配置
- 端点管理

2. 事件管理(/pages/Events):

- 事件列表查询
- 事件详情查看
- 事件筛选和过滤

3. 数据分析(/pages/EventAnalysis):

- 事件分析
- 用户行为分析
- 漏斗分析
- 趋势分析
- 来源分析

4. 系统设置(/pages/Settings):

- 用户管理
- API密钥管理
- 权限配置

5. 使用文档(/pages/Documentation):

- 快速开始
- 接入指南
- API文档
- 最佳实践

整体架构采用前后端分离,三个项目之间通过 HTTP API 进行通信。这种模块化的设计使得系统具有良好的可扩展性和可维护性。

四、测试结果

4.1、功能测试

核心功能测试结果

```
1. 事件追踪功能
   ✓ 点击事件追踪 (data-track-click)
   ✓ 浏览事件追踪 (data-track-view)
   ✓ 输入事件追踪 (data-track-input)
   ✓ 自定义事件追踪 (sendEvent)

2. SDK 初始化
   ✓ 项目注册功能
   ✓ API 端点配置
   ✓ 无效配置的错误处理

3. 用户交互事件
   ✓ 广告卡片关闭事件
   ✓ 搜索提交事件
   ✓ 社交媒体分享点击
```

4.2 测试用例详情

```
1. 广告卡片交互测试
   - 关闭按钮功能验证
   - 事件数据完整性检查
   - 时间计算准确性验证

2. 搜索功能测试
   - 空输入处理
```

```
    - 事件触发验证
    - 数据负载验证

3. 社交分享测试
    - Twitter 分享链接有效性
    - Facebook 分享链接有效性
    - 事件追踪准确性
```

4.3 性能指标分析

```
1. 事件处理性能
    - 平均事件处理时间：<50毫秒
    - 事件队列处理：<100毫秒/批次
    - 内存占用：额外增加<5MB

2. 网络影响评估
    - 数据包大小：每个事件<2KB
    - 带宽使用：通过批处理最小化
    - 请求频率：队列控制优化

3. 浏览器性能影响
    - CPU 使用率：空闲时<1%
    - 内存泄漏：未检测到
    - DOM 操作：已优化
```

4.4. 优化建议

1. 事件批处理优化
    - 实现智能批处理算法
    - 根据设备性能优化队列大小
    - 为关键事件添加优先级
2. 网络传输优化
    - 实现重试机制
    - 大数据包压缩
    - 退出事件使用 beacon API
3. 资源使用优化
    - 减少 DOM 查询次数
    - 非关键功能延迟加载

- 添加元素移除时的清理机制

```
1. 已发现问题 TBC
   - 高流量场景下事件队列溢出
   - 广告卡片移除时的内存泄漏
   - 快速触发事件时的竞态条件

2. 解决方案
   - 实现队列大小限制
   - 添加事件监听器清理
   - 为快速事件添加防抖处理
```

4.5、后续改进建议

1. 测试覆盖率提升
   - 添加端到端测试套件
   - 实现自动化性能测试
   - 增加跨浏览器兼容性测试
2. 功能增强
   - 添加离线事件存储
   - 实现实时事件验证
   - 添加自定义事件过滤
3. 文档完善
   - 补充详细的 API 文档
   - 包含性能基准测试结果
   - 添加故障排除指南

五、Demo 演示视频

演示视频

六、项目总结与反思

6.1. 目前存在的问题

在当前项目中，我们识别出了一些关键问题：

- **数据库层面**: MongoDB 的连接没有实现连接池管理和重试机制，这可能导致在高负载时出现连接失败的情况。
- **错误处理**: 目前的错误处理逻辑较为简单，缺乏统一的错误处理中间件，导致错误处理机制不够健壮，无法提供详细的错误信息。
- **SDK性能**: 事件队列的刷新机制可能导致数据的丢失，尤其在当前会话未结束时，事件未能及时上报。

6.2. 已识别的优化项

为了解决当前存在的问题，我们识别了多个优化方向：

- **服务端优化**:
  - 增加请求限流来防止过载。
  - 实现数据压缩，减少网络带宽消耗。
  - 引入缓存层以提高响应速度。
  - 完善日志记录，便于后期的故障排查。
- **SDK优化**:
  - 实现离线存储功能，确保在网络不稳定时仍能记录事件。
  - 支持批量上报，减少请求次数。
  - 添加数据压缩功能以提高传输效率。
  - 允许配置采样率，以灵活控制数据上报频率。
- **平台优化**:
  - 增加更多的数据可视化选项，提升用户体验。
  - 提供自定义报表功能，以满足不同用户需求。
  - 实现数据导出功能，方便用户处理和分析数据。

6.3. 架构演进可能性

当前的架构为 `track-sdk -> track-server -> MongoDB` 和 `track-platform` 。未来可以考虑以下演进方向：

- 引入消息队列以解耦服务，提高系统的灵活性和可扩展性。
- 使用时序数据库存储实时数据，提升数据处理效率。
- 增加实时计算能力，以支持更复杂的数据分析需求。
- 考虑微服务拆分，以提高系统的可维护性和可扩展性。

6.4. 项目反思总结

优点:

- 三端分离的架构设计清晰，便于管理和维护。
- SDK 的轻量级封装提高了集成的便利性。
- 平台功能完整，能够满足多样化的用户需求。

不足:

- 缺乏完整的测试覆盖，可能影响系统的稳定性。
- 当前的错误处理机制不够健壮，需增强。
- 性能优化空间较大，有待进一步提升。
- 缺少监控和告警机制，无法及时发现系统问题。

6.5. 未来拓展方向

1. 完善测试覆盖，确保系统的稳定性和可靠性。
2. 增强系统可用性，提升用户体验。
3. 优化性能指标，提升系统的响应速度和处理能力。
4. 丰富分析功能，以提供更深入的数据洞察和决策支持。