**生产链路前端性能埋点方案 Copy**

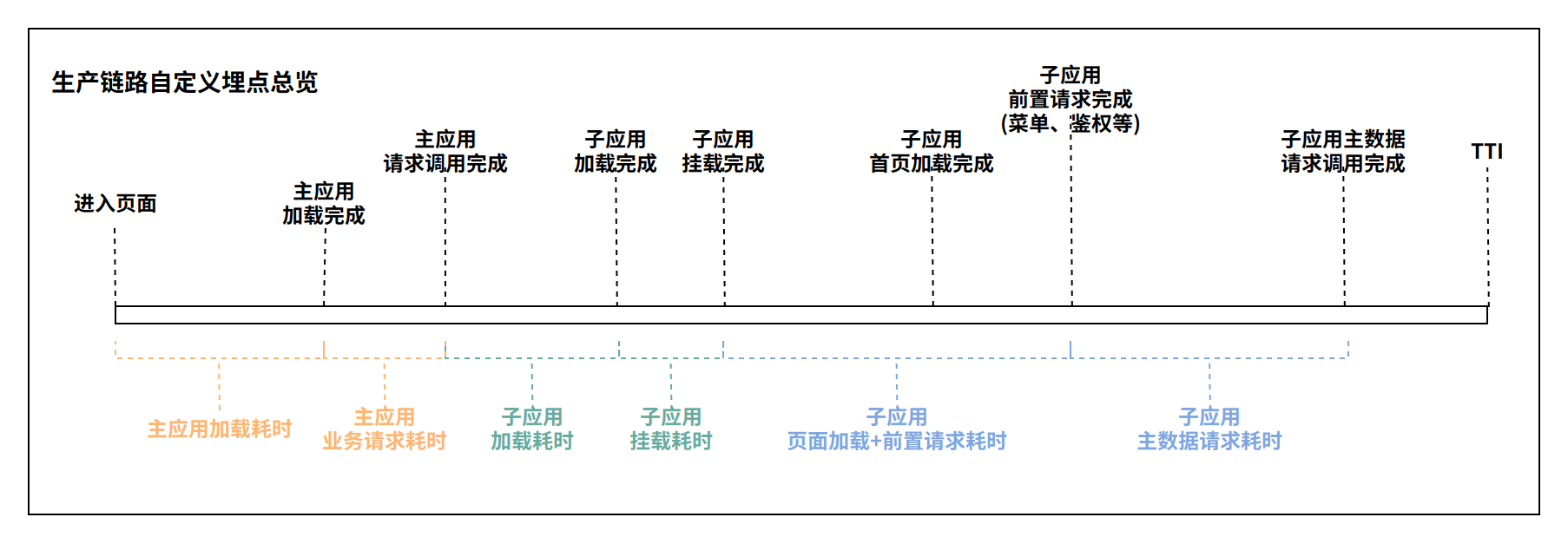
|  |
| --- |
| Owner：@Yuqin Yang@Linxiao Yang |

|  |
| --- |
| 前期埋点梳理：[生产链路前端性能&异常埋点梳理](https://bytedance.feishu.cn/docx/CZ6AduBwWoFuikx5ewechPaMnqd)  最终看板：[Slardar 生产链路-每周看板](https://slardar-us.bytedance.net/node/web_org/kanban/detail/120?oid=s_operation_fe&site_type=web&org_type=0&start_time=1676274781&end_time=1676278381&lang=zh&env=production) |

**前置调研：Slardar LCP指标有效性验证**

|  |  |  |  |
| --- | --- | --- | --- |
| **页面名称** | **主数据响应完成耗时(ms)** | **Slardar LCP(ms)** | **结论** |
| PLM 开发管理 | 9125 | 9202 | 整体符合预期，大部分LCP时间晚于主数据请求响应时间。但存在少量LCP偏小和LCP=0的情况，因此并不能直接认为LCP=列表数据已加载完成。  LCP=0的场景为：Slardar会取用户首次交互 前的最后一个 largest-contentful-paint 来作为 LCP 上报的（即最靠近用户交互的那一个），但在交互前实际上可能还没有 LCP 产生，此时的 LCP 会以我们兜底的 0 进行上报，在数据处理时会忽略LCP=0的值。  本地和线上都会偶现LCP偏小或早于数据返回响应的数据，其原因可能为：用户首次交互较早，此时已经产生LCP（例如浏览器视窗较小，search-form元素较多时），这时候上报的LCP可能会在数据响应之前。 |
| 8985 | 9062 |
| PLM 选品池 | 8366 | **4338** |
| 8188 | 9830 |
| 版房 | 7252 | 13055 |
| 6909 | 10818 |

**埋点方案**



**子应用路由跳转埋点@Linxiao Yang**

**场景**

目前dbmp系统中存在四种路由跳转场景：

* 首屏路由跳转
* 子应用 -> 子应用（未加载）
* 子应用 -> 子应用（已加载）
* 子应用内部路由跳转

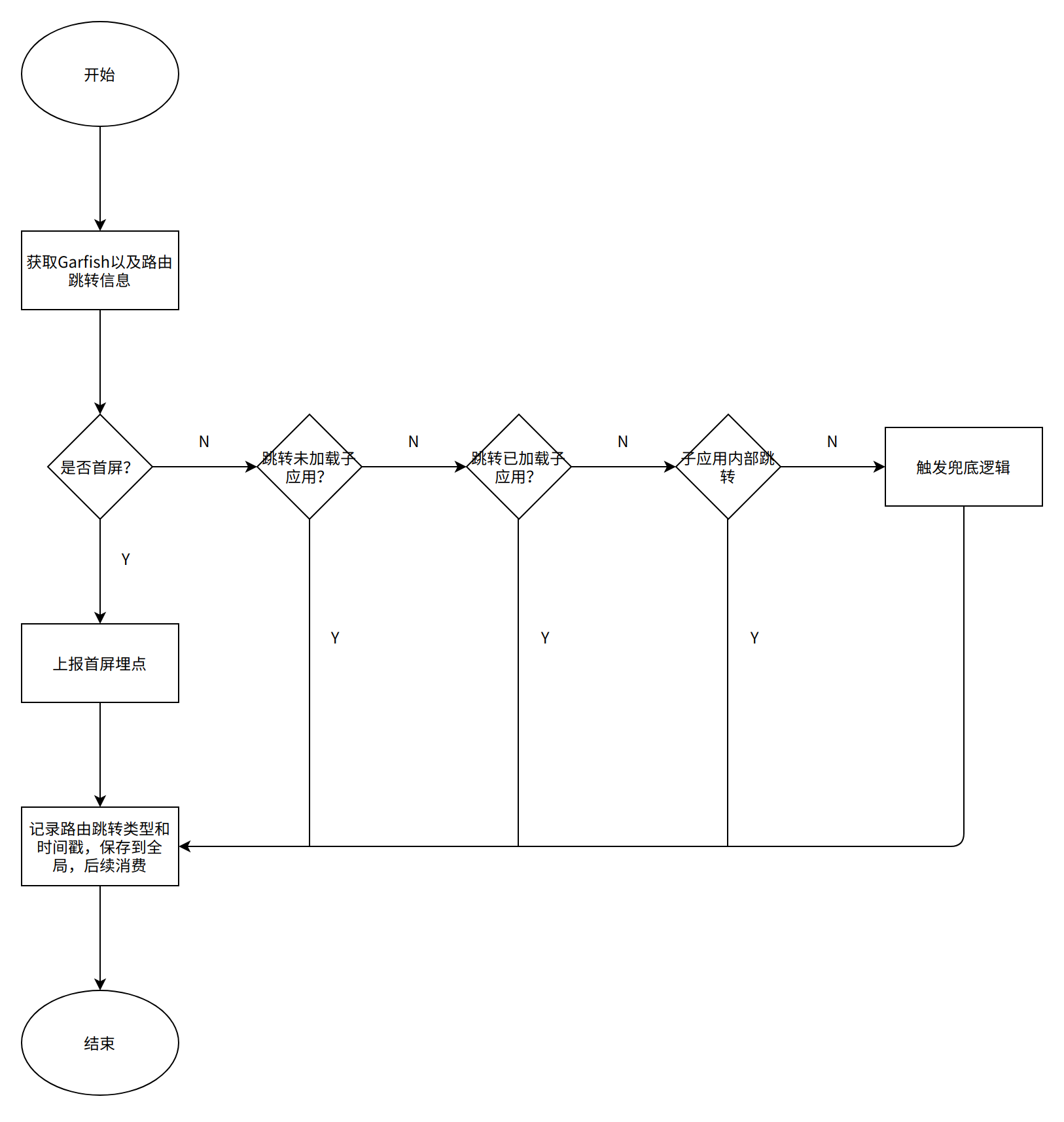
**影响范围**

主应用

**实现方案**

借助[window.Garfish.router.beforeEach](https://www.garfishjs.org/api/router#garfishrouterbeforeeach)对每种路由场景进行埋点或者时间戳的记录，后续消费

**流程图**



**代码实现**

|  |
| --- |
| TypeScript // 路由钩子 export function addRouterHook() {  window.Garfish.router.beforeEach((to, from, next) => {  const { appInfos, activeApps, cacheApps } = window.Garfish;  // 获取目标路由对应的应用信息  const toAppInfo = getAppInfo(appInfos, to.path);  // 获取当前路由对应的应用信息  const fromAppInfo = getAppInfo(appInfos, from.path);  // 路由跳转类型  let routeJumpType: RouteJumpType;  if (!activeApps.length) {  // 主应用在加载子应用之前的耗时  routeJumpType = RouteJumpType.Init;  const duration = performance.now();  // 上报首屏之前耗时，也算是主应用初始化耗时  slardarInstance.sendEvent?.({  name: SlardarCustomEvent.MainAppBeforeLoadSubAppUseTime,  metrics: {  duration,  },  categories: {  url: to.path,  app: toAppInfo.name,  },  });  console.log(  `${SlardarCustomEvent.MainAppBeforeLoadSubAppUseTime} :>> `,  duration,  );  } else if (activeApps.length && !cacheApps[toAppInfo.name]) {  // 子应用跳转未加载的子应用  routeJumpType = RouteJumpType.ExternalUnLoaded;  } else if (  activeApps.length &&  cacheApps[toAppInfo.name] &&  toAppInfo.name !== fromAppInfo.name  ) {  // 子应用跳转已加载的子应用  routeJumpType = RouteJumpType.ExternalLoaded;  } else if (toAppInfo.name === fromAppInfo.name) {  // 子应用内部跳转  routeJumpType = RouteJumpType.Internal;  } else {  // 兜底  routeJumpType = RouteJumpType.Unknown;  }  // 记录时间戳，后续消费  setRouteJumpToGlobal({  type: routeJumpType,  timestamp: Date.now(),  });  next();  }); } |

**看板**

集成到[Slardar 生产链路-每周看板](https://slardar-us.bytedance.net/node/web_org/kanban/detail/120?oid=s_operation_fe&site_type=web&org_type=0&start_time=1676274781&end_time=1676278381&lang=zh&env=production)

**子应用加载/挂载埋点@Linxiao Yang**

**场景**

* 子应用首次/非首次加载，首次和非首次加载时间相差比较大，为了数据更加有区分度和准确，所以区分上报
* 子应用首次/非首次挂载，同上
* 子应用加载失败，单独上报可以直观感知每个子应用加载失败情况

**影响范围**

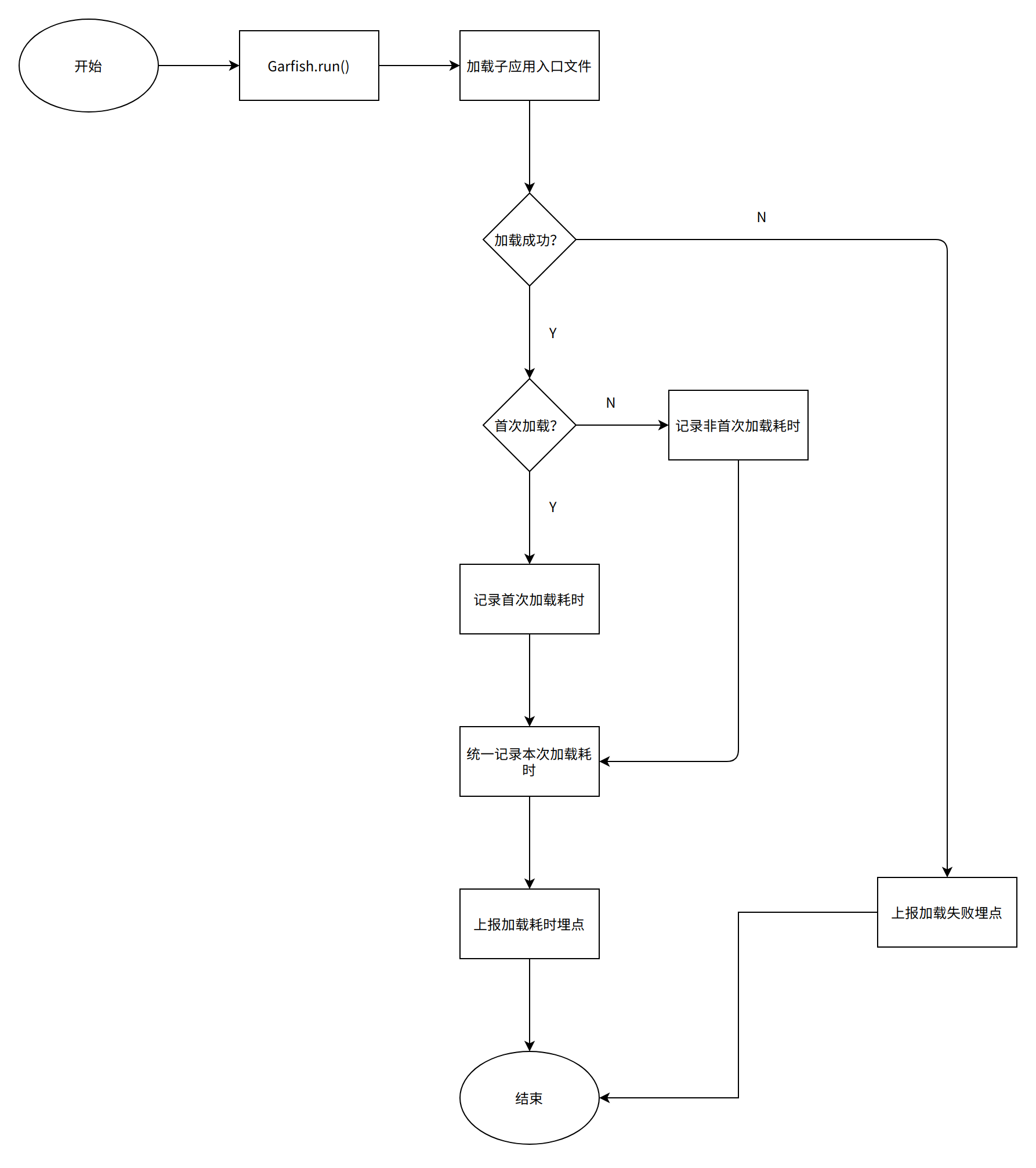
主应用

**实现方案**

借助[Garfish.run(options?: interfaces.Options): Garfish](https://www.garfishjs.org/api/run)进行配置

**流程图**

子应用加载和挂载实现差不多，这里只展示加载部分。



**代码实现**

|  |
| --- |
| TypeScript // 这里写成map，主要考虑到出现加载多个子应用的情况 const map: Record<string, Record<string, any>> = {};  const runConfig: Parameters<typeof window.Garfish.run>[0] = {  beforeLoad(appInfo) {  // 记录当前加载时间，在afterLoad中使用  map[appInfo.name] = map[appInfo.name] || {};  map[appInfo.name].loadStartTime = Date.now();  // 记录是否首次加载  map[appInfo.name].isFirstLoad = !window.Garfish.cacheApps[appInfo.name];  },  afterLoad(appInfo) {  const { loadStartTime, isFirstLoad } = map[appInfo.name];  // 加载子应用耗时  const loadSubAppUseTime = Date.now() - loadStartTime;  // 上报加载子应用的耗时  slardarInstance.sendEvent?.({  name: SlardarCustomEvent.LoadSubAppUseTime,  metrics: {  // 每次都上传duration可以统一记录加载耗时，不区分首次和非首次  duration: loadSubAppUseTime,  // 区分首次和非首次上报  [getDurationMetricsKey(isFirstLoad)]: loadSubAppUseTime,  },  categories: {  // 根据子应用分类上报  app: appInfo.name,  },  });  console.log(  `${SlardarCustomEvent.LoadSubAppUseTime} :>> `,  loadSubAppUseTime,  );  },  errorLoadApp(error, appInfo) {  // 记录加载失败次数  slardarInstance.sendEvent?.({  name: SlardarCustomEvent.ErrorLoadSubApp,  metrics: {  count: 1,  },  categories: {  // 根据子应用分类上报  app: appInfo.name,  },  });  // 重新把error抛出去给全局捕获  throw error;  },  beforeMount(appInfo) {  // 记录当前挂载开始时间，在afterMount中使用  map[appInfo.name] = map[appInfo.name] || {};  map[appInfo.name].mountStartTime = Date.now();  },  afterMount(appInfo) {  // 挂载子应用耗时  const { mountStartTime, isFirstLoad } = map[appInfo.name];  const mountSubAppUseTime = Date.now() - mountStartTime;  // 上报挂载子应用的耗时  slardarInstance.sendEvent?.({  name: SlardarCustomEvent.MountSubAppUseTime,  metrics: {  // 每次都上传duration可以统一记录加载耗时，不区分首次和非首次  duration: mountSubAppUseTime,  // 区分首次和非首次上报  [getDurationMetricsKey(isFirstLoad)]: mountSubAppUseTime,  },  categories: {  // 根据子应用分类上报  app: appInfo.name,  },  });  console.log(  `${SlardarCustomEvent.MountSubAppUseTime} :>> `,  mountSubAppUseTime,  );  }, };  export default runConfig; |

**看板**

集成到[Slardar 生产链路-每周看板](https://slardar-us.bytedance.net/node/web_org/kanban/detail/120?oid=s_operation_fe&site_type=web&org_type=0&start_time=1676274781&end_time=1676278381&lang=zh&env=production)

**主数据响应埋点 @Yuqin Yang**

**背景**

根据前置对Slardar LCP指标有效性的调研，浏览器采集的LCP并不能保证用户看到带有完整数据的页面，当用户在数据响应前进行交互时，LCP数据会偏小。为了准确记录页面初始化数据完成响应的时间、还原用户真实使用情况，我们需要对各子应用页面的主数据响应时间进行埋点。

**场景**

* 首屏初始化数据响应时间：从打开首屏到所有初始数据完成响应的时间，使用performance.now() 获取；
* 子应用初始化数据响应时间：从子应用跳转到所有初始数据完成响应的时间，使用 当前timestamp - 子应用跳转时记录的timestamp 获取；

**影响范围**

* feature/common-fetch 包
* 各子应用入口文件

**实现方案**

**方案1：自动化配置❌:**

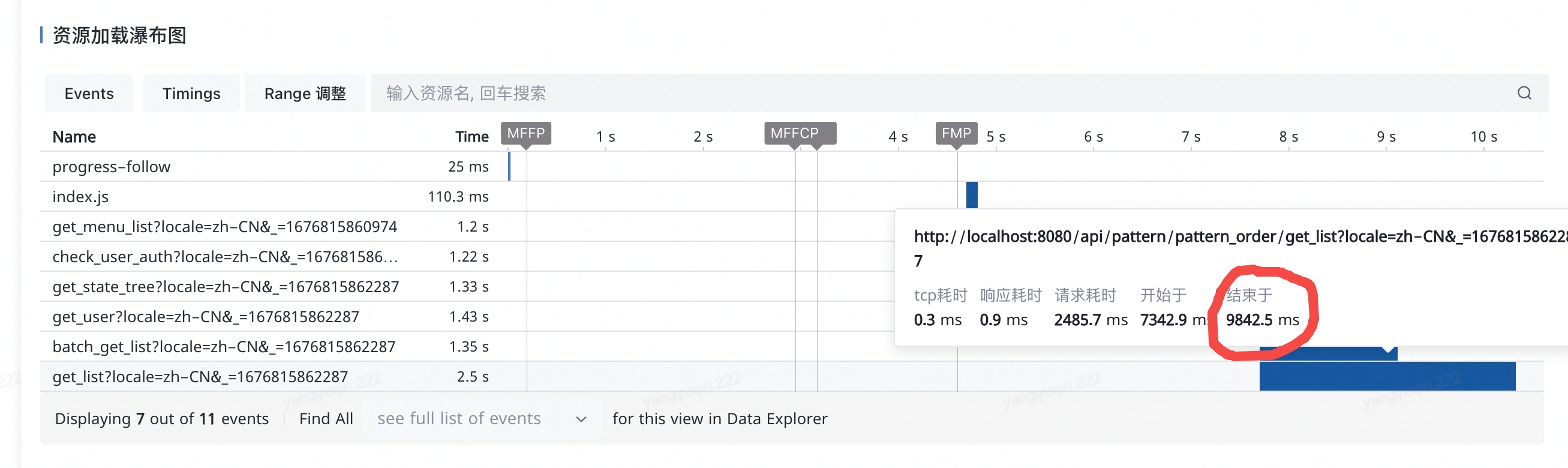
基于webpack loader实现，各子应用维护RequestTrack配置文件，通过loader向代码中插入slardar上报方法。已实现demo，验证方案可行。

|  |
| --- |
| TypeScript export const RequestTracks = [  {  filePath: 'src/pages/overall/index.tsx', // 请求调用文件  requestName: 'MGetSample', // 请求方法名  }, ]; |

这个方案虽实现了自动注入、无需改动源代码；但各子应用需要维护配置，请求变更时需要手动更新配置，这增加了开发同学在新增、修改请求时的额外成本，因此不选用此方案。同样的，也不选用babel-loader进行代码配置化注入的方案，代码配置化的方式没有严格约束，loader对源代码改造的负面影响是无法预估的。

**方案2：Slardar 数据整合❌**

基于Slardar收集的数据，直接收集指定接口的响应时间。在Slardar收集了首屏整个session的全链路数据，并记录了每个请求的响应结束时间，我们预期通过获取某个指定请求的首次响应结束时间作为埋点时间。



咨询Slardar Oncall后确认此方案不可行，有以下两个原因：

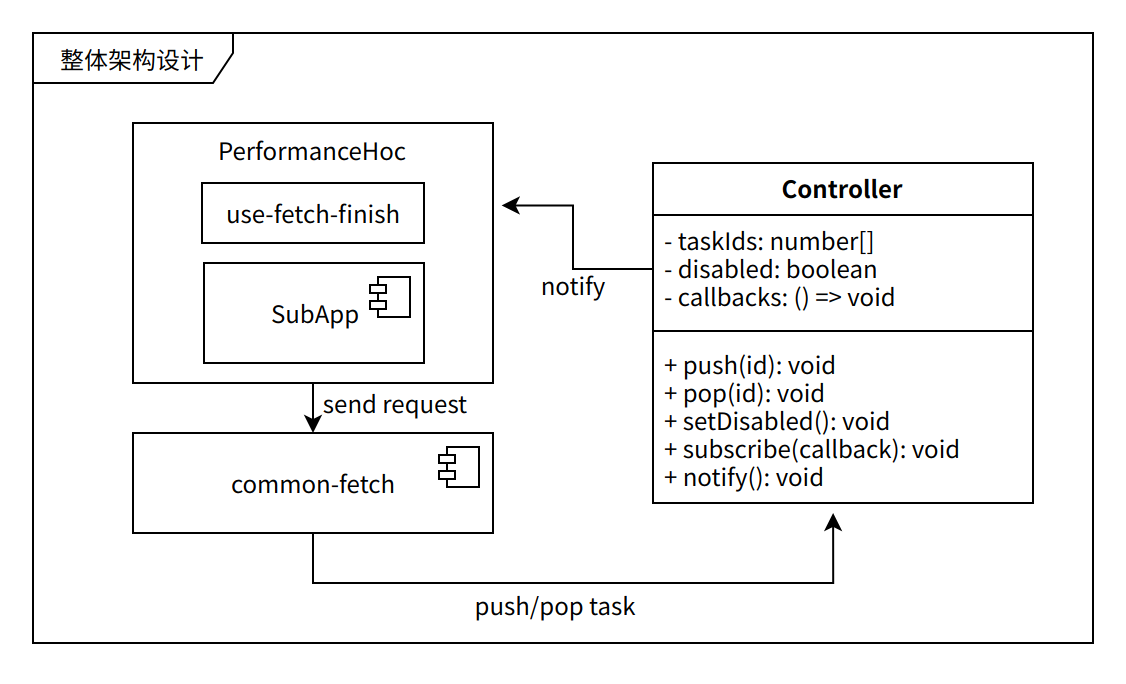
1. Slardar 看板中暂不支持 respondEnd（响应结束时间）指标的查询，无法在看板中进行统计呈现；
2. 一个 session 链路可能包含多个数据请求，我们无法分辨哪一次请求来自于首屏初始化；

**方案3：封装高阶组件，订阅首屏初始化请求 ✅**

|  |
| --- |
| **感谢@Geliang Yan提出的方案与demo实现** |

整体架构设计如下图所示👇

* Controller类作为中间层控制器，负责维护任务列表。
* common-fetch库在任务执行前向队列中push task，在异步任务结束后pop task，由于我们仅关注请求是否完成、并不关注请求顺序，因此直接pop队尾元素即可。当队列中所有元素均被推出后，控制器Controller将触发 notify发布方法，依次执行callbacks回调函数。
* PerformanceHoc以高阶组件的形式实现，在需要埋点的子应用入口组件外层使用，并传入slardarSendEvent方法用于埋点上报。在PerformanceHoc的useEffect中调用useFetchFinish，此Hook负责定义subscribe方法，在subscribe中实现埋点上报逻辑。



**代码实现**

PerformanceHoc实现👇

|  |
| --- |
| TypeScript type PerformanceHocProps {  slardarSendEvent: (params: SendEventProps) => void; } export const PerformanceHoc = (Component: any) => ({ slardarSendEvent }: PerformanceHocProps) => {  const addFetchFinishTrack = useFetchFinish({ slardarSendEvent });  useEffect(() => {  addFetchFinishTrack(); // 终止fetch计入  }, []);   return <Component {...props} />; }; |

useFetchFinish 实现👇

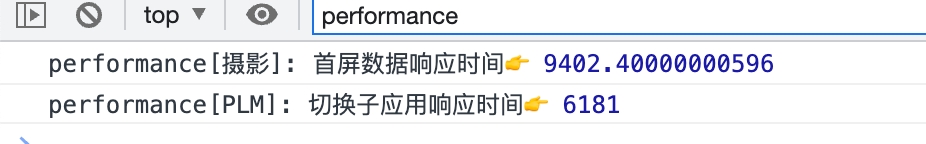
|  |
| --- |
| TypeScript const useFetchFinish = ({ slardarSendEvent }: Props) => {  // 全局的单例Controller  const ref = useRef<Controller | undefined>(undefined);  // 首次初始化时设置subscribe回调函数  useEffect(() => {  ref.current = controllerPool.get();  ref?.current?.subscribe(() => {  if (routeJumpType === 'init') {  // 上报首屏数据响应时间...（省略）  } else {  // 切换子应用响应时间..（省略）  }  controllerPool.clear();  });  }, []);   return () => {  controllerPool.setDisabled();  }; }; |

common-fetch改造👇

|  |
| --- |
| TypeScript // request interceptors this.axios.interceptors.request.use((config: SAxiosRequestConfig) => {  const controller = controllerPool.get();  const now = Date.now(); // 采用timestamp作为本次task id  if (controller) {  controller.push(now); // 新任务推入  }  // 其他原有逻辑 }); // response interceptors this.axios.interceptors.response.use(  (response: SAxiosResponse) => {  // 其他原有逻辑  controller && controller.pop(now); // 任务完成，id推出  },  (error: SAxiosError) => {  // 其他原有逻辑  controller && controller.pop(now); // 任务完成，id推出  }, ); |

**上线计划**

已完成demo开发+验证，计划本周完成代码与看板上线。



**打印异常埋点 @Yuqin Yang**

**场景**

打印异常埋点需要覆盖前端打印场景，目前各个系统均使用feature/op-component封装的打印组件op-print，因此在op-print中提供通用埋点方法即可。

**影响范围**

* feature/op-component/op-print组件
* 提供统一props方法，各子应用自行接入

**实现方案**

<Print>组件新增 commonReport 属性，由外部组件提供上报方法，外部组件无需关注打印组件内部的埋点逻辑。

|  |
| --- |
| TypeScript export interface CommonReportParams {  level: string;  content: string; }  interface PrintCompProp {  // 【优先使用】通用的日志上报方法, 外部仅提供log上报方法，不感知上报时间  commonReport?: (params: CommonReportParams) => void; } |

埋点位置：

|  |
| --- |
| TypeScript // 开始打印前上报日志，用于记录打印操作调用次数 if (startPrint && isAllLoad) {  commonReport?.({  level: 'info',  content: 'readyToPrint',  });  }  // 监听onPrintError方法，用于上报打印异常和异常信息 onPrintError: (errorLocation, error) => {  commonReport?.({  level: 'warn',  content: `errorPrint: errorLocation=${errorLocation}, error=${JSON.stringify(error)}`,  }); }, |

**上线计划**

* ~~PLM-产品管理 2月23日~~
* ~~版房 2月23日~~
* 样品管理 3月2日

**首屏re-render耗时埋点（delay）**

|  |
| --- |
| 2.16补充：Profiler若放在顶层页面组件，会触发大量render update，这些update的耗时差距较大，难以判断实际页面的重渲染实际。且Profiler的插入可能会带来额外的性能开销，本期暂时搁置，未来再调研其他方案。 |

**影响范围：**各子应用首页（PLM、版房、摄影、样品），以及部分PLM需要关注性能的页面。

**实现方案：**使用[React Profiler API](https://zh-hans.reactjs.org/docs/profiler.html#gatsby-focus-wrapper) 提供的<Profiler>组件包裹需监控的子应用父级组件，并封装通用的onRender回调方法。以页面为维度，建议分析各菜单首页的组件，详情页本期暂不做支持。<Profiler>需写在被分析的组件外部，有以下两种实现方式：

1. 手动（或写个简单脚本）添加<Profiler>组件；【优点：简单、成本低、灵活性高、可指定页面分析；缺点：需要手动添加】
2. 借助webpack-loader，自动为各个主路由入口文件插入<Profiler>组件；【优点：自动、统一、可扩展性高；缺点：需要开发成本、针对PLM内部各菜单首页需要遍历路由来添加组件】
3. 能否直接写在app.tsx里？特别是PLM子应用

|  |
| --- |
| TypeScript // 封装通用的onRender方法，上报埋点 const onCompReRenderTrack = (url: string, info: {  id: string,  phase: 'mount' | 'update',  actualDuration: number, }) => {  // 上报重渲染耗时埋点  if (phase === 'update') {  slardar.sendEvent?.({  name: 'comp\_re\_render\_duration',  metrics: {  duration: info.actualDuration,  },  categories: {  url,  },  });  } } |

**看板搭建 @Yuqin Yang**

[Slardar 生产链路-每周看板](https://slardar-us.bytedance.net/node/web_org/kanban/detail/120?oid=s_operation_fe&site_type=web&org_type=0&start_time=1676274781&end_time=1676278381&lang=zh&env=production)