**生产链路前端性能&异常埋点梳理 Copy**

|  |
| --- |
| Owner：@Yuqin Yang |

**目标**

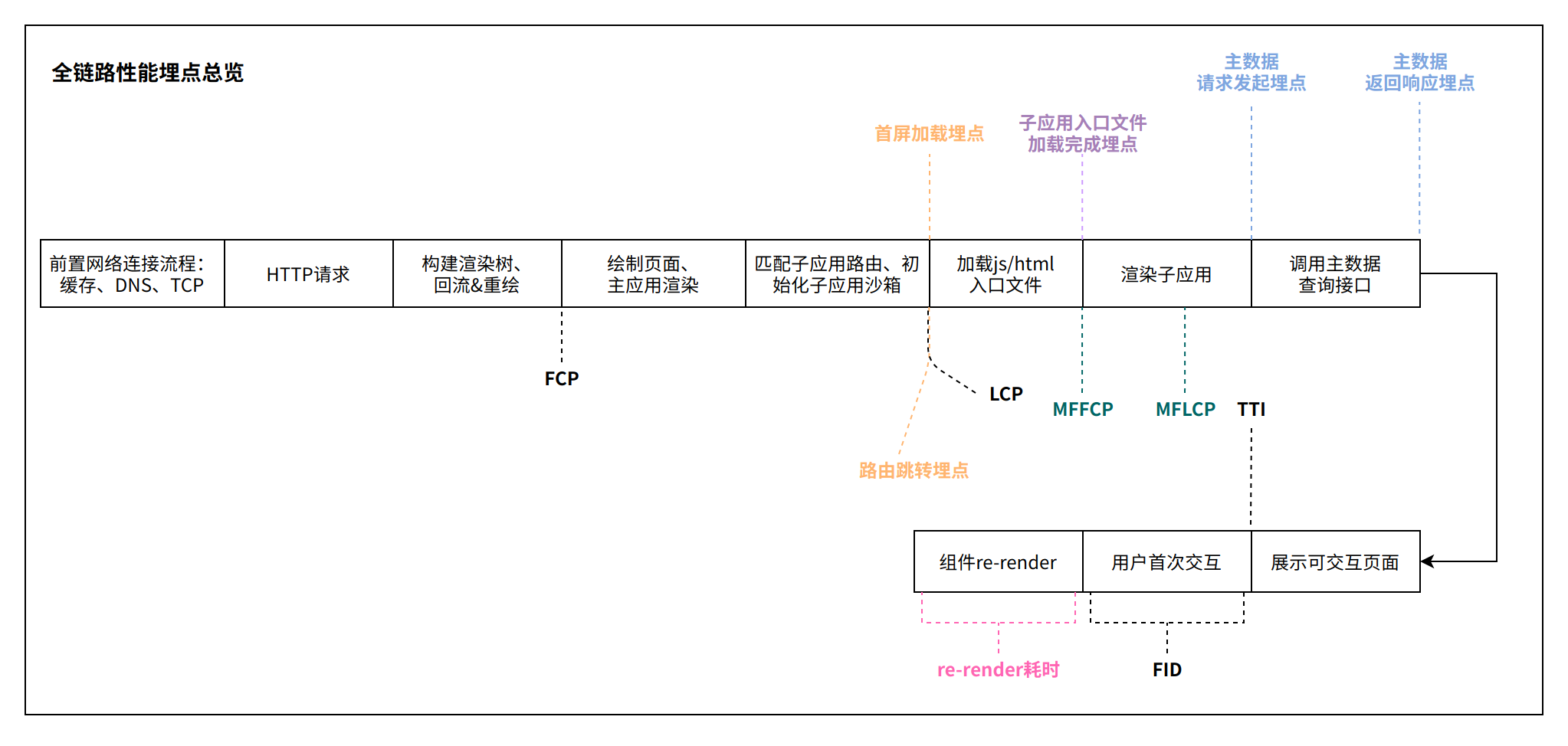
目前前端性能监控仅依赖于Slardar自动化收集的数据，而Slardar提供的FCP、LCP性能指标较为宽泛，无法准确反映各个子应用的性能情况。因此，我们对子应用中需要关注、且与用户体验相关的性能埋点进行梳理，并给出这些埋点的收集方案，基于埋点数据可以进一步设计更为详细的性能指标。生产链路和其他方向的子应用均可以参考这些指标，发现并解决系统中存在的性能问题。

整体节奏：

* 埋点梳理
* 埋点&指标设计
* 看板建设
* 代码实现

**性能埋点**

**系统流程**



**埋点 / 指标梳理**

|  |
| --- |
| **核心耗时获取方法：Performance.now()**  返回值表示为从[time origin](https://developer.mozilla.org/zh-CN/docs/Web/API/DOMHighResTimeStamp#the_time_origin)之后（navigationStart后）到当前调用时经过的时间，不同于Date类型，这个返回值以浮点数的形式表示时间，精度更高。 |

|  |  |  |  |
| --- | --- | --- | --- |
| **埋点名称** | **描述** | **实现方案**  以下Slardar截图来自于02-01至02-04开发管理-全部页面的75分位数据 | **优先级** |
| FCP | 记录从网页开始加载到页面任意部分内容渲染在屏幕上的时间点。 | Slardar提供 | 低 |
| LCP | 记录网页可视区内“绘制面积”最大的元素开始呈现在屏幕上的时间点。使用网页Performance观测性能时发现：在LCP时间点并没有显示表格组件，而是展示子应用loading页。从业务视角看，LCP似乎并不能反映实际页面主元素的绘制时间。   |  |  | | --- | --- | | 版房👇： | PLM👇： | | Slardar提供 | 中 |
| 首屏子应用加载埋点 | 记录子应用开始加载的时间点。 | 【手动上报】  两个埋点均在匹配子应用路由后进行，可以在主应用监听路由变化，并设置全局变量(或全局状态) route\_count用于记录子路由跳转次数。   * route\_count = 1时，使用performance.now()上报具体页面的首屏子应用加载埋点； * route\_count > 1时，使用performance.now()上报具体页面的子应用路由跳转埋点； | 高 |
| 子应用路由跳转埋点 | 记录监听到子应用路由跳转的时间点。 | 高 |
| 子应用入口文件加载完成埋点 | 记录子应用入口文件（目前生产链路子应用均是js入口）加载完成的时间点。若此时间在整条链路中占用时间过长，则需要进一步进行包体积优化。 | 【手动上报】  ~~在子应用useEffect(() => {}, [])中使用performance.now()上报具体页面的首屏子应用入口加载完成埋点。~~  待讨论：由于useEffect Hook中子应用已挂载完成，与实际入口文件加载完成时间存在一些误差（约100～200ms），有没有更好的埋点位置？   * Garfish.run的options参数中提供了[beforeMount](https://www.garfishjs.org/api/run#beforemount)方法，子应用资源准备完成、运行时环境初始化完成、准备开始渲染子应用 DOM 树时被触发，可以在此进行埋点。 | 中 |
| MFFCP | 记录微前端子应用首次渲染内容（文本、图片、带背景图的内容）的时间点，在性能统计指标中，从微前端子应用开始加载的时间点到 MFFCP 的时间点这段时间可以被视为微前端子应用无内容时间，也就是说在用户访问微前端子应用的过程中，MFFP 时间点之前，用户看到的都是没有任何实际内容的屏幕。 | Slardar提供 |  |
| MFLCP | 记录微前端子应用最大内容绘画的时间点，在性能统计指标中，从用户开始访问微前端子应用的时间点到 MFLCP 的时间点这段时间可以被视为微前端子应用最大内容绘画。 | Slardar提供 |  |
| 主数据请求发起埋点 | 记录主数据调用前的时间点。 | 【手动上报】  封装通用接口，在接口调用前后，手动上报具体页面的请求发起和返回时间。 | 高 |
| 主数据返回响应埋点 | 记录主数据成功返回的时间点，各子系统对主数据的需求不同，可能调用多个后端接口。利用（返回响应埋点-请求发起埋点）可以得到主数据请求时间。 |
| TTI | 记录从页面开始加载到页面的主要资源已加载并能够响应用户输入的时间。 | Slardar提供 | 高 |
| FID | 记录用户第一次与网站交互到浏览器实际响应交互的时间间隔。   |  | | --- | | ❓为什么偶尔会出现较长时间的FID❓    若用户在可交互时间前与系统交互，此时主线程可能被其他任务占用，需要等待其他任务完成后才执行交互响应。这也是为什么不建议使用95或90分位数据的原因，我们需要排除掉一些偶发的异常数据。 | | Slardar提供 | 中 |
| CLS | 记录网页布局在加载期间的偏移量，是对页面的整个生命周期中发生的每一次意外布局变化的最大布局变化得分的度量。 | Slardar提供 | 中 |
| 子应用首屏re-render耗时埋点 | 记录子应用首屏父级组件每重渲染的耗时情况。 | 【手动上报】  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。    onRender方法中phase="update"时，上报actualDuration作为当前页面的重渲染耗时。 | 高 |

**异常埋点**

**业务请求异常埋点**

将commonFetch进行一层封装（或直接在commonFetch内部改造），当响应数据base\_resp中状态码不为0时手动上报业务异常，并将状态码设置为categoriestype以便于数据统计分析。目前质量监控专项已经对业务异常进行了log上报，后续我们可以对这些log进行统计处理和展示。 [Slander-WMS业务异常Log](https://slardar-us.bytedance.net/node/web/data_search?env=ONLINE&bid=s_operation_wms&lang=zh&start_time=1675047987&end_time=1675652787&site_type=web&subregion=row&layout=normal&ev_type=log&filter_id=e1ca33299450107b1bdfd95971b8fd47&granularity=auto&time_shortcut=last_seven_days)

|  |
| --- |
| TypeScript // web-ajax.ts this.axios.interceptors.response.use(  (response: SAxiosResponse) => {  // const errNo = response.data?.err\_no;  const errNo = response.data?.base\_resp?.code;  if (errNo != null && errNo !== 0) {  response?.config?.slardar?.sendLog?.({  content: `base\_resp.code 不为 0, url=${  response?.config?.url ?? ''  }, msg=${response?.data?.base\_resp?.message}, logid=${  response?.headers?.['x-tt-logid']  }, request data=${JSON.stringify(response?.config?.data ?? {})}`,  level: 'info',  });  if (this.loginErrors.includes(errNo)) {  return this.testLoginError(response);  }  return this.handleError(response);  } else {  return this.handleSuccess(response);  }  }, } |

**图片异常埋点**

**💡图片上传异常**

BytedUploader提供了uploader.on('error', info => {})方法来监听上传过程中的错误情况，用户可以从info对象中获取errorCode与errorMessage。目前图片上传存在许多异常情况[上传2.0 错误码](https://bytedance.feishu.cn/wiki/wikcngvbbCahwgP8xxjjwA2X1Ta#ysPFNm) ，异常情况的上报可以分为两类：

1. 通用上报，将errorCode作为category type上报所有异常情况。
2. 对部分常见情况（如token错误、token过期、请求次数过多等，需要讨论确认）进行特殊事件上报，可以清晰地了解需要关注的异常数据，目前版房特殊上报了token异常的数据@Langxi Guan。

待确认@Ludwig Zeng：能否在arco-s图片上传组件中接入异常埋点，各子应用传入slardar实例（或上报方法），由组件内部统一进行埋点上报。

**💡图片展示异常**

[Slardar Web 图片监控接入指南](https://bytedance.feishu.cn/wiki/wikcnN15CP54IiTAnH5p3kEgVMf)

[Slardar-PLM图片监控](https://slardar-us.bytedance.net/node/web/image?env=ONLINE&bid=s_operation_main&lang=zh&start_time=1675072436&end_time=1675331636&site_type=web&subregion=row&layout=normal&activeTab=0)

无需在代码中手动上报，直接接入imagx-Slardar插件即可完成监控，目前质量专项小组已完成所有子应用的图片监控接入。

**打印异常埋点**

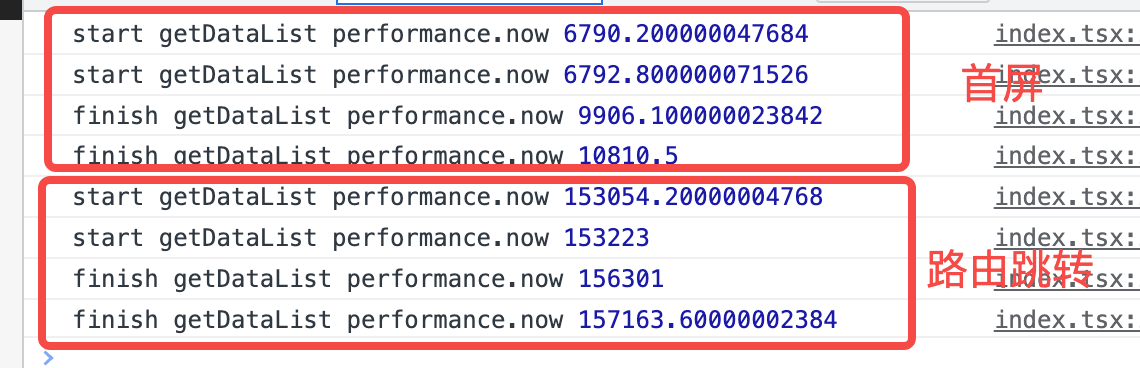
目前生产链路共有2种打印实现方式，分别是浏览器原生打印和封装的react-to-print打印，两种实现方式的异常处理方式不同。

* window.print()：不提供异常监听方法，但打印报错时会将报错信息打印至控制台，可以利用try…catch捕获异常，catch error时将异常上报至Slardar。
* react-to-print：提供异常监听方法 onPrintError ，可以在此方法中直接上报打印异常及异常步骤。

|  |
| --- |
| TypeScript **onPrintError**?:   (**errorLocation: "onBeforeGetContent" | "onBeforePrint" | "print"**, error: Error)   => void; |

**2.6遗留问题**

* 使用Performance.now()记录时间点时，若从子应用路由跳转而非首屏，这个时间点是否会很长？
* **结论：**Performance.now()的起点是navigationStart，只能记录首屏到当前时间的间隔，无法区分起点是首屏还是路由跳转。



* **更新埋点上报方式：**路由监听中设置的全局变量除route\_count外，额外记录路由跳转时间route\_jump\_time。判断全局变量route\_count = 1时，直接上报Performance.now()；route\_count > 1时，上报 Performance.now() - route\_jump\_time 作为路由跳转后的实际时间点。
* 是否可以直接使用MFLCP来计算子应用最大元素渲染的时间点？

|  |  |
| --- | --- |
|  |  |

* **LCP/MFLCP的准确性：**咨询Slardar Oncall后得知若loading占用整屏确实会影响LCP（MFLCP）的计算，重新检查Element后确认<LoadingPage />组件并未占用整屏DOM，猜测浏览器Performance的LCP计算有问题，在开启lighthouse后可以看到LCP和screenshot是符合预期的（如下左图）。同时在Slardar数据探索中查看[选品池列表页首屏性能指标](https://slardar-us.bytedance.net/node/web/data_search?env=ONLINE&bid=s_operation_plm&lang=zh&start_time=1676020164&end_time=1676106564&site_type=web&subregion=row&layout=normal&ev_type=view&filter_id=e1ca33299450107b1bdfd95971b8fd47&granularity=60&time_shortcut=lay_one_day&nav_id=1bb61b5c5f77665a82926a78d427a8f5)，可以看到MFLCP先于LCP触发，同样符合预期（如下右图）。

|  |  |
| --- | --- |
|  |  |

* **结论**：根据进一步的咨询和确认，认为Slardar上报的LCP/MFLCP数据是（暂时）符合预期的，在本双月时间较少的情况下，建议直接使用LCP作为子应用最大元素渲染时间的主要性能参考指标
* 进一步设计数据请求的埋点方案，尽量减少现有代码的改造成本。 [生产链路前端性能埋点设计](https://bytedance.feishu.cn/docx/AqVxdBKe9oj02GxRTSdcaR9pnFb)

**参考文档**

[前端监控指标 & 告警规则](https://bytedance.feishu.cn/wiki/wikcnwzKzy4U1xH3gk9mZy05kre)

[前端性能埋点规范-slardar篇](https://bytedance.feishu.cn/wiki/wikcn2TpVNc07ezaEOWe66lpv1c)

[首屏埋点方案](https://bytedance.feishu.cn/wiki/wikcnIchsP5qT3L7DRINTSP1ixh#ACTCxB)

[交互响应性能之FID - 掘金](https://juejin.cn/post/6860132156400730120)

[React Profiler 的使用 - 掘金](https://juejin.cn/post/7008337341634854942)

[性能指标计算原理](https://bytedance.feishu.cn/wiki/wikcny7kzXdTeEQpCNLeyBq7rNe#dq92Xd)

[性能监控 FAQ](https://bytedance.feishu.cn/wiki/wikcn2mlirzBEHXUrmWhix4L3oJ)