1.0 需求背景

当前背景：质量大盘是统计版本开发和线上运营数据的展示看板，通过记录各个版本的bug数，修复率，线上更新次数等数据反映开发测试的工作质量，使用方式是通过搜索项目、切换项目、切换版本来展示指定版本数据图表和整体趋势图表。
目前主要存在以下问题：

1. 特定版本数据和趋势数据耦合
2. 部分接口获取趋势数据时涉及重复和嵌套的IO操作，导致查询性能差。
   原业务逻辑图：

用户请求
│
▼
[查询接口（/getVersionData）]
│
├── 1. 查询特定版本数据（实时）
│ ├── 读取数据库 → 版本基础信息（Bug数、修复率等）
│ └── 读取外部系统 → 线上Bug数据（jira/小亮数据）
│
└── 2. 构造趋势数据（历史版本聚合）
│
├── 2.1 遍历所有相关版本（循环）
│ │
│ ├── 对每个版本：
│ │ ├── 查询数据1 → Bug数（IO操作1）
│ │ ├── 查询数据2 → 修复数（IO操作2）
│ │ ├── 查询数据3 → 修复截止时间（IO操作3）
│ │ └── 特殊逻辑处理（如顺序依赖计算）
│ │
│ └── （问题：循环嵌套导致O(N)次IO，无法并发）
│
├── 2.2 聚合数据
│ ├── 计算趋势字段（如“全部Bug率”，依赖版本数据）
│ └── 生成图表结构（JSON/Array）
│
└── （问题：耦合版本数据，计算耗时 >10s）

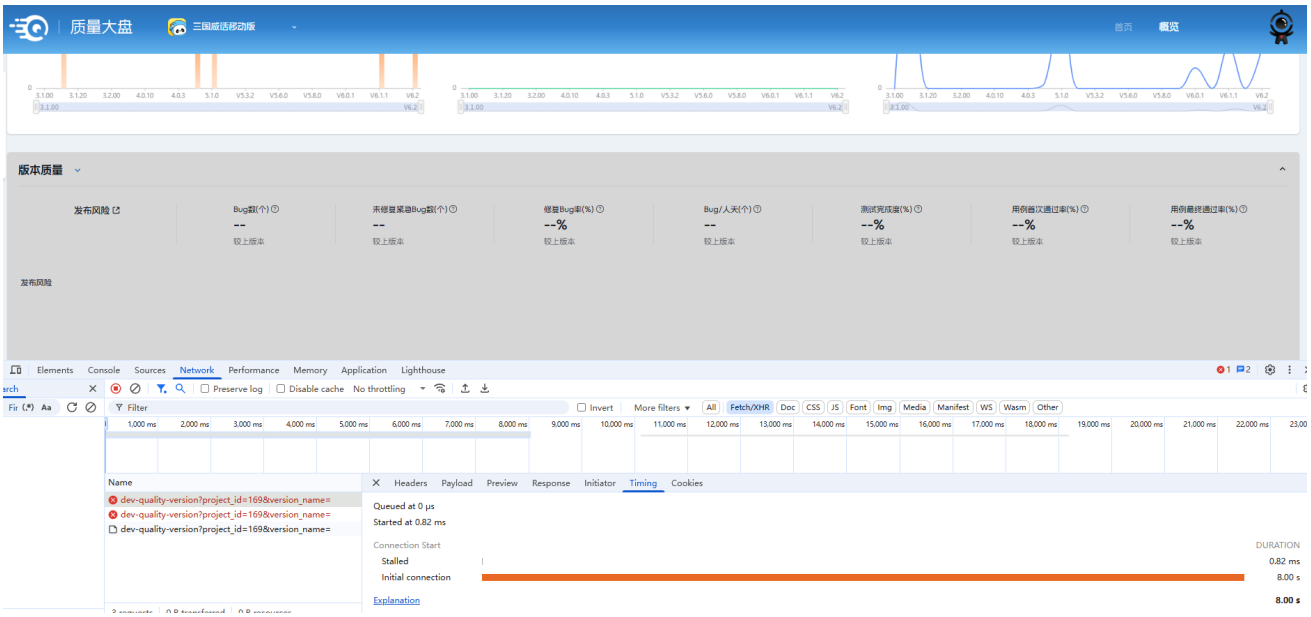
│
▼
[返回结果]
├── 版本数据（快速）
└── 趋势数据（慢速，阻塞主流程）

2.0 方案优化

2.0.1 原设计方案

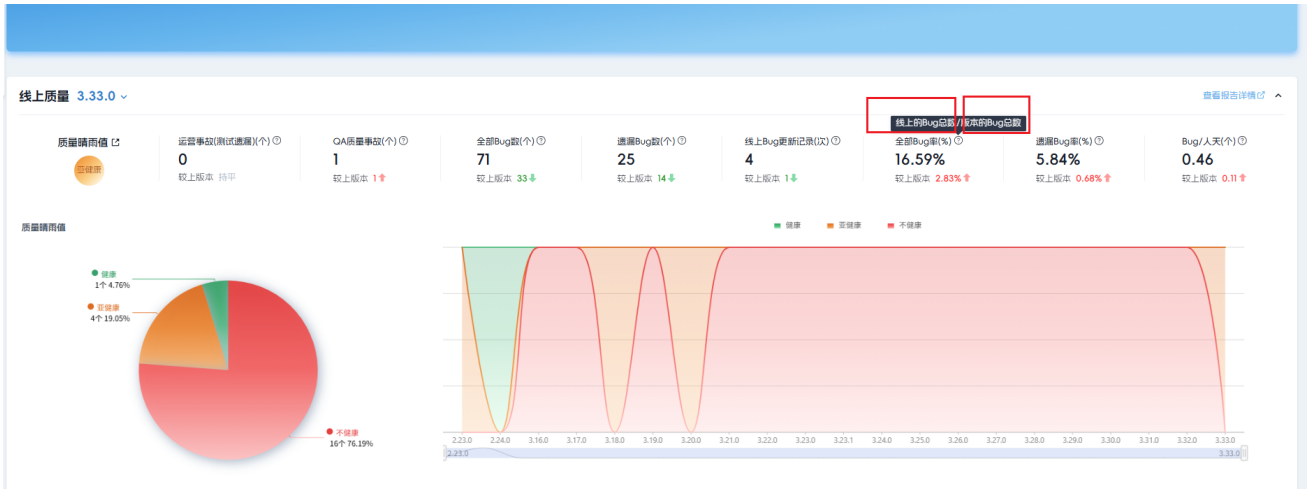
1. 同一个接口返回特定版本数据和趋势数据，切换版本时同时查询版本数据和趋势数据，趋势数据规模较大，最坏情况下会大于10s，阻塞数据返回（前端接口8s重试机制）。

阻塞性接口实例



2. 构造趋势数据时循环查询含多层嵌套关系：遍历版本名查询对应的线上数据（jira/小亮相关信息）。
3. 构造趋势数据时循环包含各种顺序关系的特殊处理，无法使用并发优化。
4. 线上质量数据和版本质量数据在构造时有数据依赖，强耦合关系。例如线上质量的“全部Bug率(%)”字段在计算时依赖版本质量的数据。

质量大盘数据计算参考



2.0.2 优化设计方案

针对目前的存在的问题基于影响范围规划了以下两个阶段的方案。两个阶段的方案相互独立，且阶段二的方案可从阶段一平滑过渡。

2.0.2.0 阶段一

影响范围：仅后端业务逻辑。
阶段一的设计以外部接口不变为原则，面向IO相关的耦合和重复逻辑进行优化，前端页面的接口无需变动，涉及的测试范围为线上质量面板，版本质量面板，首页卡片面板等明显卡顿的部分功能。

1. 质量大盘的趋势数据接口通常基于用户已发送的报告数据，优化报告数据的IO操作：如果有多种报告关联的数据，使用多线程同时完成IO，当前的报告规模数据规模为几千条，除“线上质量复盘报告”外可一次完成50ms以内的IO（线上质量报告关联协同数据数据需要多一次IO开销）。以二层嵌套，20个版本的项目为例，优化前仅报告查询耗时平均约为20ms*20=400ms，优化后 预估平均耗时70ms，提升80%性能。
   风险评估：低，当前某一特定类型的报告的增长速度约为300条/年，只有数据规模十分庞大时约2.5w条才会出现卡顿。（以100MB/s带宽，每份报告2KB，数据库走索引，同时需要两种类型的报告数据为参考）

```
N= (100MB/s *1s)/2KB/2 = (100,000KB)/2KB/2=25,000
```

2. 趋势数据中包含多个实时数据，优化实时数据的IO操作：
   循环中每个版本都会重复发起IO操作，尝试以版本列表为参数进行IO操作，如果多个不同的IO数据如果性能较差使用多线程优化。

```
原方案：

for 版本 in 版本列表:

// 耗时部分
{
        data1 = 读取数据(版本)

        data2 = 读取数据(版本)

        data3 = 读取数据(版本)
}

处理数据(data)
```

```
优化方案：

版本列表 = 获取所有版本()

// 耗时部分
{
        数据列表 = 批量读取数据(版本列表) （如果多个数据耗时大需要增加多线程）
}

for data in 数据列表:

处理数据(data)
```

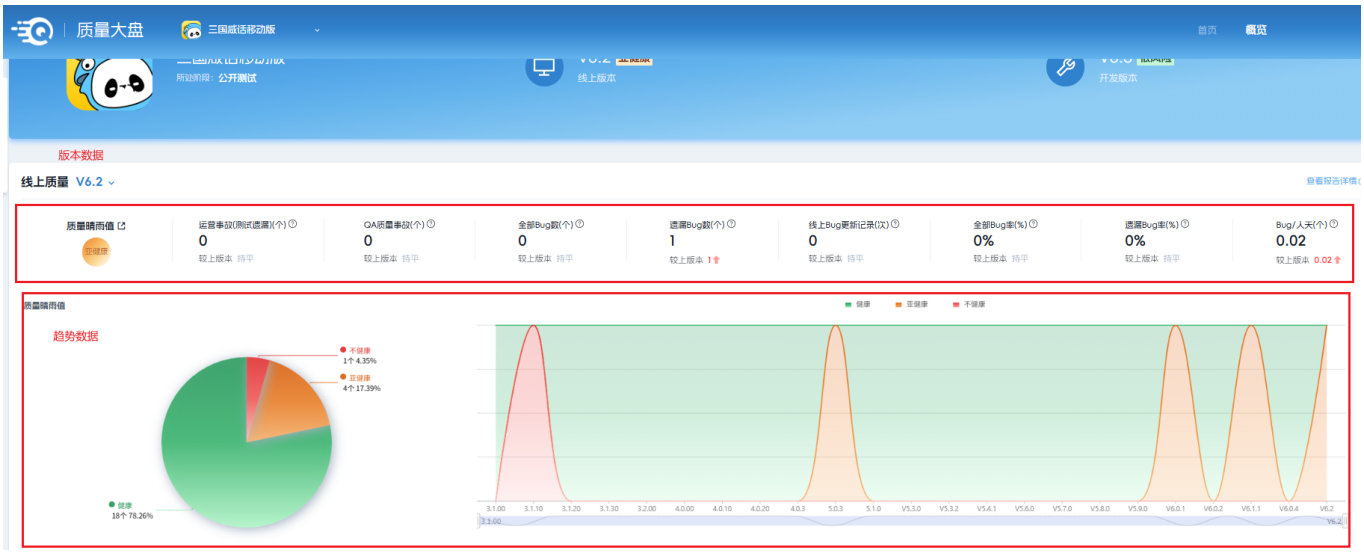
风险评估：中，数据的内容部分为小亮接口，按照版本列表批量查询数据需要外部支持。如外部不支持，可以实现部分优化。

2.0.2.1 阶段二

影响范围：80%以上前后端接口

阶段二的设计方案在阶段一的基础上拆分数据接口。

1.原数据结构包含同时包含版本数据和趋势数据，不仅在功能上不相关，而且前端会在查询时对整个区域进行重新渲染，增加无关的展示效果和耗时，预估减少特定接口90%查询时间。



优化方案：趋势数据和版本数据用多个接口返回，前端可以分开调用，先显示版本数据，趋势数据用加载中的状态，等接口返回后再显示。

- **版本数据**：优先加载，展示核心指标（如Bug数、修复率）。
- **趋势数据**：

```
- 初始请求返回`loading`占位符，异步轮询任务状态。

- 支持用户手动刷新，避免频繁自动请求。
```

优点：后端去除包含多处特殊处理且查询效率低下的循环逻辑，提升查询效率和可维护性，前端可以实现平滑展示。

风险：中，变更的范围大，测试范围大，前后端接口和前端组件都要重新处理数据。

2.0.2.3 其他（时效性）

平台性质：数据看板，趋势展示，实时性要求不高。
当前平台查询的jira/小亮数据每次进行实时IO，没有使用任何缓存。而平台的数据源是隔一个小时进行更新，结合平台实时性要求低的性质可在针对多个需要外部IO计算的数据进行缓存，优化查询效率。
例：

1. **“全部Bug率”字段解耦**
   - **关键点**：需验证离线计算的“全部Bug率”与实时版本数据计算结果的一致性。
   - **建议**：
     - 初期可保留双链路计算，通过日志对比验证，逐步迁移至独立数据源。

2. **分层数据模型**
    - **预计算层设计**：
        - 定时任务需具备重试机制与监控告警，避免计算中断导致数据缺失。
        - 聚合层存储时，建议增加时间分区字段（如`week_start`），便于快速查询。

风险：中，变更的范围大，测试范围大，且会引入新的表结构和中间件和缓存数据不一致问题。

3.0 总结

3.0.0 方案评估

3.0.1 阶段一优化方案评估

1. **IO批量查询与多线程优化**
    - **有效性**：通过批量查询替代循环嵌套IO，结合多线程处理，理论上可显著减少IO耗时（如从400ms降至70ms）。

3.0.2 阶段二优化方案评估

1. **接口拆分与解耦**
    - **优势**：
        - 版本数据与趋势数据分离，前端可优先展示核心信息，提升用户体验。
        - 后端逻辑简化，便于维护和扩展。
    - **风险应对**：
        - **前后端协同**：建议制定接口变更的兼容性策略（如版本控制、灰度发布）。
        - **测试覆盖**：针对拆分后的接口设计自动化测试用例，确保数据一致性。

3.0.3 风险与收益再平衡

风险点	缓解措施	阶段
外部接口不支持批量查询	改用多线程并发单次请求，限制最大并发数（如10线程）。	阶段一
阶段二前后端协作成本高	采用契约测试（如Pact）确保接口兼容性，分模块逐步迁移。	阶段二

3.0.1 最终结论

当前方案具备可行性，**阶段一优化可优先落地**，风险可控且收益明确；**阶段二需谨慎推进**，建议分模块迭代并加强自动化测试。
整体目标：通过解耦与性能优化，将版本数据接口响应时间稳定控制在1s内，趋势数据首次加载时间降低至3s以下。