项目申请书

项目名称：实现在 OpenGemini Go/Java 客户端中无缝集成 OpenTelemetry

项目主导师:徐业

申请人：陈欢

邮箱：xiangyuyu_2024@qq.com

日期：2025 年 6 月 1 日

1. 项目背景

1.1 项目基本需求

OpenGmini 作为开源时序数据库，需提升可观测性以增强用户对数据采集、处理流程的监控能力。OpenTelemetry 是云原生领域标准的观测性框架，本项目目标：

1. Go 客户端集成：在 OpenGemini Go 客户端中添加 OpenTelemetry 支持，实现链路追踪、指标采集与日志聚合。
2. Java 客户端集成：在 Java 客户端中完成相同功能，确保跨语言一致性。
3. 标准化输出：数据格式遵循 OpenTelemetry 协议(OTLP)，支持对接 Prometheus、Jaeger 等主流观测工具。

1.2 项目相关仓库

OpenGemini Go 客户端仓库：https://github.com/openGemini/opengemini-client-go
OpenGemini Java 客户端仓库：https://github.com/openGemini/opengemini-client-java
OpenTelemetry 官方文档：https://opentelemetry.io/zh/docs/

2. 技术方法及可行性

2.1 核心技术

1. OpenTelemetry SDK

1. Go 端：

1) 创建一个 Interceptor 拦截器接口，提供 QueryBefore,QueryAfter,WriteBefore,WriteAfter 方法。
2) Client 继承该 Interceptor，方便后续实现多个拦截器。
3) 创建一个结构体 OtelClient 集成 otel 客户端并实现 Interceptor 接口。

2. Java 端：

利用 OpenTelemetry 的 Java API 和 SDK 进行配置与追踪，通过拦截器或装饰器模式在客户端方法调用前后插入追踪逻辑，借助 Span 来记录操作的开始、结束及相关属性，同时配置合适的 Exporter 将追踪数据发送到后端进行分析和展示。

2. 协议与传输

采用 OTLP gRPC/HTTP 协议，将数据发送至 OpenTelemetry Collector，支持自定义 Endpoint 配置。

3. 兼容性设计

保留原有客户端逻辑，通过配置开关控制 OpenTelemetry 功能的启用/禁用，避免影响现有业务。

2.2 可行性分析

1. **技术储备：**

熟悉 Go/Java 开发（参考简历项目经验），曾参与开源项目 OpenGemini 学习，了解其客户端架构。

掌握 Docker 容器化部署（简历知识储备），可用于搭建 OpenTelemetry 测试环境。

2. **参考资源：**

OpenTelemetry 官方示例：https://github.com/lightstep/opentelemetry-examples

类似项目实践：InfluxDB 客户端集成 OpenTelemetry 方案

3. 项目实现细节

3.1 Go 客户端集成

1. 初始化流程

```
import(
    "context"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
)

//配置 OTLP gRPC exporter
exporter, _ :=
otlptracegrpc.New(context.Background(),otlptracegrpc.WithInsecure())
otel.SetTracerProvider(otel.NewTracerProvider(otel.WithBatcher(exporter)))
```

2. 关键链路追踪

追踪 Connect() 、 Query() 等核心方法，记录数据库操作耗时与错误信息。

添加自定义标签（如 db.instance=opengemini 、 operation=query ）。

3.2 Java 客户端集成

1. 依赖引入

```
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-api</artifactId>
    <version>1.20.0</version>
</dependency>
```

2. 自动埋点

使用 OpenTelemetry Java Agent 自动拦截客户端方法，生成 Span 数据。
手动埋点示例：

```
Tracer tracer = TracerProvider.getDefault().getTracer("opengemini-java-client");
Span span = tracer.spanBuilder("query").startSpan();
try {
    // 执行数据库查询
} finally {
    span.end();
}
```

3.3 数据验证与输出

集成后通过 OpenTelemetry Collector 将数据转发至 Jaeger，验证链路完整性；
在Prometheus中配置查询规则，监控客户端指标（如 opengemini_client_requests_total ）。

4. 项目规划

4.1 第一阶段（07 月 01 日-07 月 21 日）

- 调研 OpenGemini Go/Java 客户端代码结构，确定集成切入点；
- 搭建 OpenTelemetry 本地测试环境（Collector + Jaeger + Prometheus）。

4.2 第二阶段（07 月 22 日-09 月 1 日）

- 完成 Go 客户端 OpenTelemetry SDK 集成与单元测试；
- 实现 Java 客户端自动埋点与手动埋点双模式。

4.3 第三阶段（09 月 2 日-09 月 30 日）

- 编写集成文档与使用指南，提交 Pull Request 至 OpenGemini 官方仓库；
- 参与开源社区反馈，优化集成方案。

5. 项目风险评估与应对措施

5.1 技术风险

- **OpenTelemetry 版本兼容性**：不同版本的 OpenTelemetry SDK 可能存在 API 变更，导致集成代码需要调整。
- **应对措施**：提前关注 OpenTelemetry 官方更新日志，选择稳定版本进行集成，预留测试周期进行兼容性验证。
- **性能影响**：集成 OpenTelemetry 可能对客户端性能产生一定影响。
- **应对措施**：进行性能测试，优化埋点逻辑，确保集成后的客户端性能满足业务需求。

5.2 社区风险

- **开源社区反馈延迟**：提交 Pull Request 后，可能面临社区反馈延迟，影响项目进度。
- **应对措施**：提前与 OpenGemini 社区沟通，了解 Pull Request 处理流程，预留足够时间进行代码审查与修改。

6. 期望

1.提升 OpenGemini 客户端的可观测性，降低用户监控成本；
2.积累开源项目贡献经验，熟悉 CNCF 生态工具链协作流程。