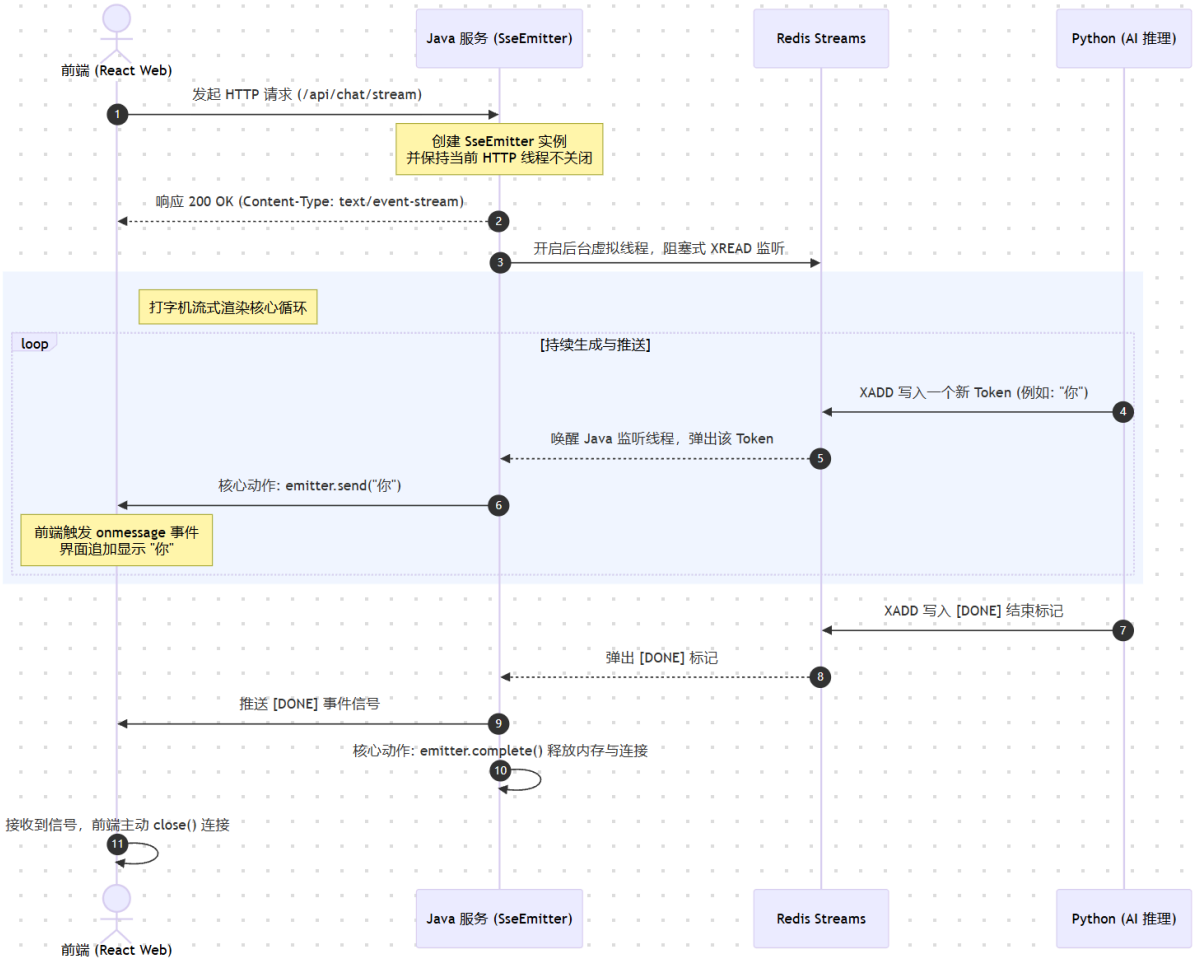
AI 智能问答架构实施规范

1. 架构总览

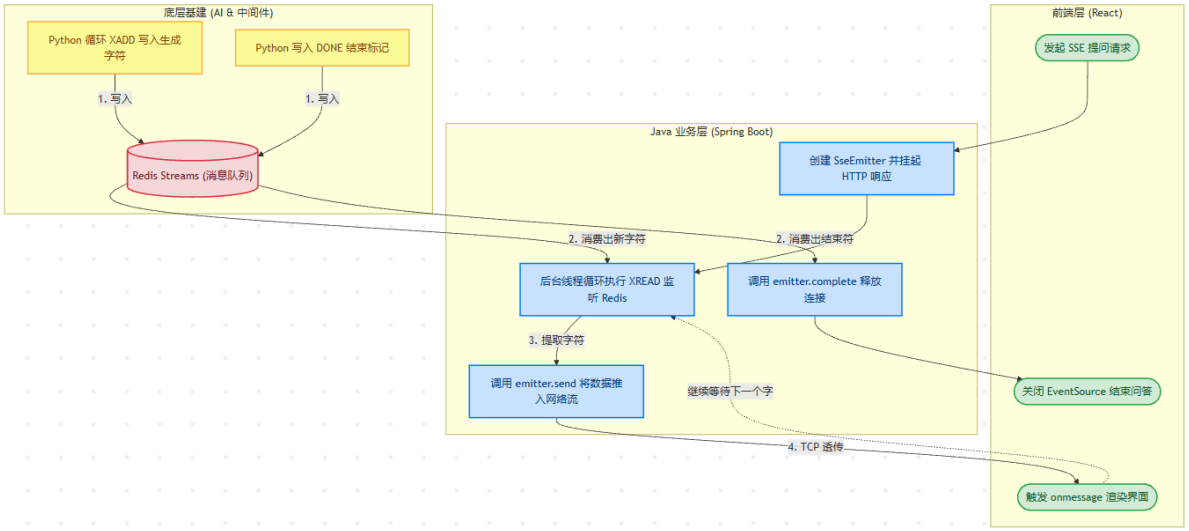
本规范定义了前端（React 19）、Java 业务线与 Python AI 引擎之间实时对话流（Streaming）的通信标准。
核心目标是实现**高可靠、无状态、可观测**的流式透传，确保大模型的打字机输出在复杂的分布式网络环境下不丢字、不断层。

核心架构原则：

- **无状态路由：** 用户的长连接可安全落至任意 Java 节点。
- **可靠消息总线：** 采用 **Redis Streams** 作为跨语言消息通信基座，支持游标读取与消息持久化。
- **生命周期闭环：** 建立严格的“前端重连补偿 + 服务端心跳保活 + 异常强制释放”机制。



3. 通信详解



3.1 展现层 (React 19 Frontend)

- **连接建立：** 使用 `@microsoft/fetch-event-source` 库发起 POST 请求建立 SSE 连接。请求 Header 必须注入全局唯一的 `X-Trace-Id` 及用户鉴权 Token。
- **防乱序与断点续传：**
  - 客户端需维护一个 `last-event-id` (或 `chunk_index`)。
  - 若网络异常断开，前端利用指数退避算法自动发起重连，并在请求中携带该索引，要求服务端从断点处继续下发。
- **全量补偿 (Fallback)：** 若 SSE 彻底断开且多次重连失败，前端调用 Java 提供的 `GET /api/chat/history/{SessionID}` 接口拉取当前完整文本进行画面覆盖，保障最终一致性。

3.2 业务接入层 (Java Spring Boot)

Java 节点在此链路中扮演“安全网关”与“流转发器”的角色，严禁执行耗时计算阻塞连接。

- **连接注册与挂起：**
  - 验证前端传入的 `SessionID` 是否隶属于当前发起请求的 `UserID` (防流窃听)。
  - 创建 `SseEmitter` (建议超时时间设为 5-10 分钟)，并将其存入节点本地内存 (如 `ConcurrentHashMap`)。
- **基于 Redis Streams 的消费：**

- 针对每个合法的 `SessionID`，启动轻量级线程（推荐 Java 21 虚拟线程）执行 `XREAD BLOCK 15000 STREAMS stream:chat:{SessionID} $`。
- 解析读到的消息，调用 `emitter.send()` 下发给前端。

- **心跳保活 (Heartbeat)：**
  - 后台运行定时任务，每 15 秒向所有活跃的 `SseEmitter` 发送注释类型的 Ping 帧：`event: ping\ndata: {}\n\n`。维持网关（如 Nginx）链路活跃。
- **强制生命周期清理：**
  - 必须在 `SseEmitter` 的 `onCompletion`，`onTimeout`，`onError` 回调中执行资源回收：移除内存引用、中断 XREAD 线程。
  - 读取到 Python 发来的 `[DONE]` 标识后，调用 `emitter.complete()`，并主动执行 `DEL stream:chat:{SessionID}` 释放 Redis 内存。

3.3 智能引擎层 (Python FastAPI)

Python 层无需维护网络长连接，作为"无状态的生产者"专注于大模型推理。

- **数据投递格式：** 每次大模型 yield 出新的字符，立即调用 Redis 客户端执行 `XADD`。写入 Payload 结构须严格统一：JSON

```
{
    "trace_id": "透传的前端 X-Trace-Id",
    "chunk_index": 12,        // 严格递增序号，供前端校验
    "content": "生成的片段",
    "is_end": false           // 是否为结束帧
}
```

- **生命周期终结声明：** 当推理正常结束，或因外部模型 API 崩溃导致中断时，Python 侧**必须**向 Redis 写入一条 `is_end: true` 的消息作为"墓碑消息"。确保 Java 侧能够收到信号并优雅释放资源。
- **兜底过期策略：** 在创建 Redis Stream Key 时，强制挂载 `EXPIRE 3600`（1小时过期）指令，防止极端异常下产生僵尸流数据。

---

4. 全链路可观测性 (Observability)

- **TraceID 贯穿：** 从用户点击发送的第一刻起，`X-Trace-Id` 必须贯穿**前端 → Java MDC → Python Logger → Redis Message** 的全生命周期。
- 出现断流、响应慢等问题时，运维团队可通过 ELK 或 SkyWalking 直接检索该 ID，精确定位性能瓶颈是发生在大模型推理端、Redis 传输层还是网络网关层。