提升流式开发效率与易用性：Kitex/Hertz 为大模型应用保驾护航

分享人：王宇轩

字节跳动服务框架团队研发工程师

目录

01 大模型应用架构概览

背景、Chat 场景流式链路

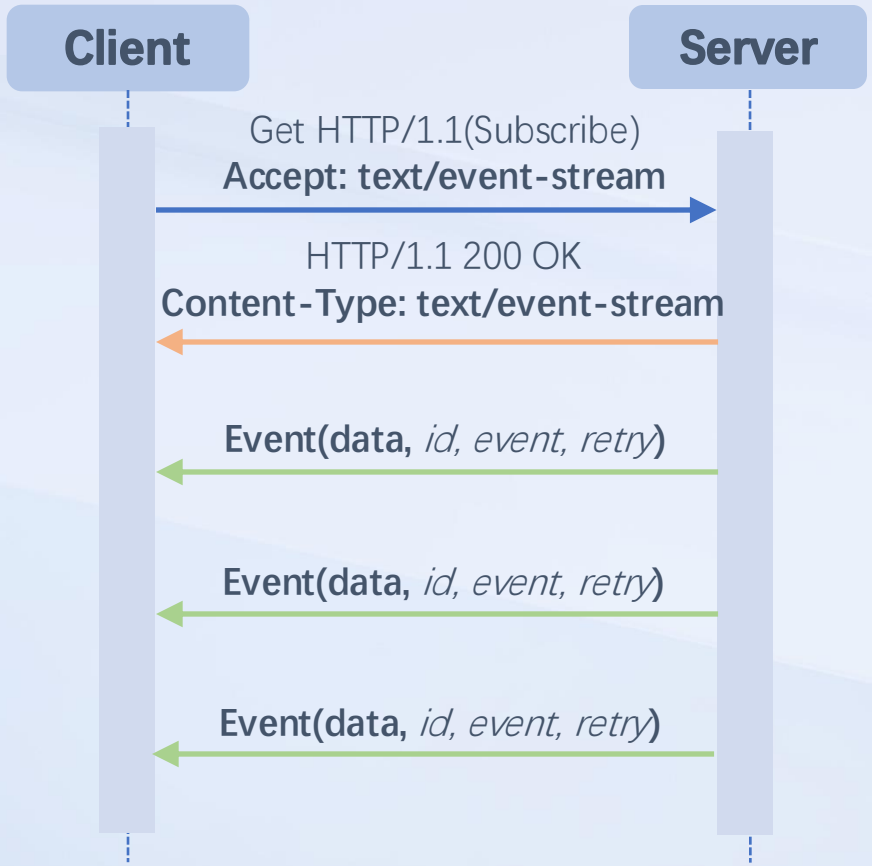
背景 | 大模型应用 & 流式交互

大模型应用蓬勃发展：Chat 类型产品领航

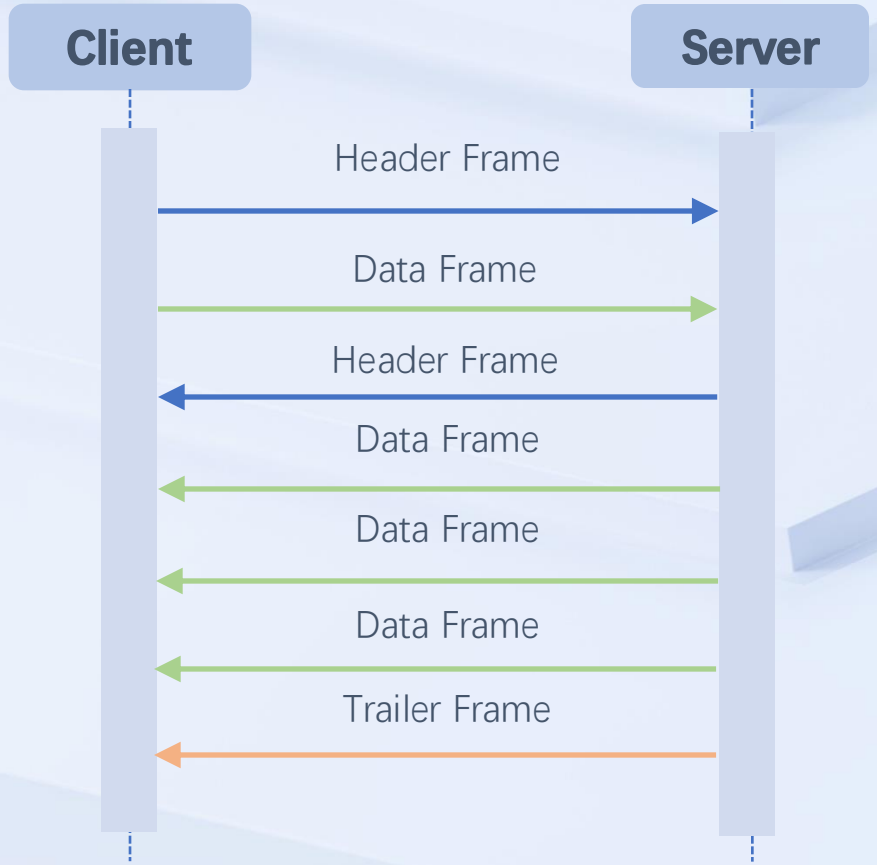
流式交互深入人心：一问多答交互模式

流式能力 | Hertz SSE & Kitex Streaming

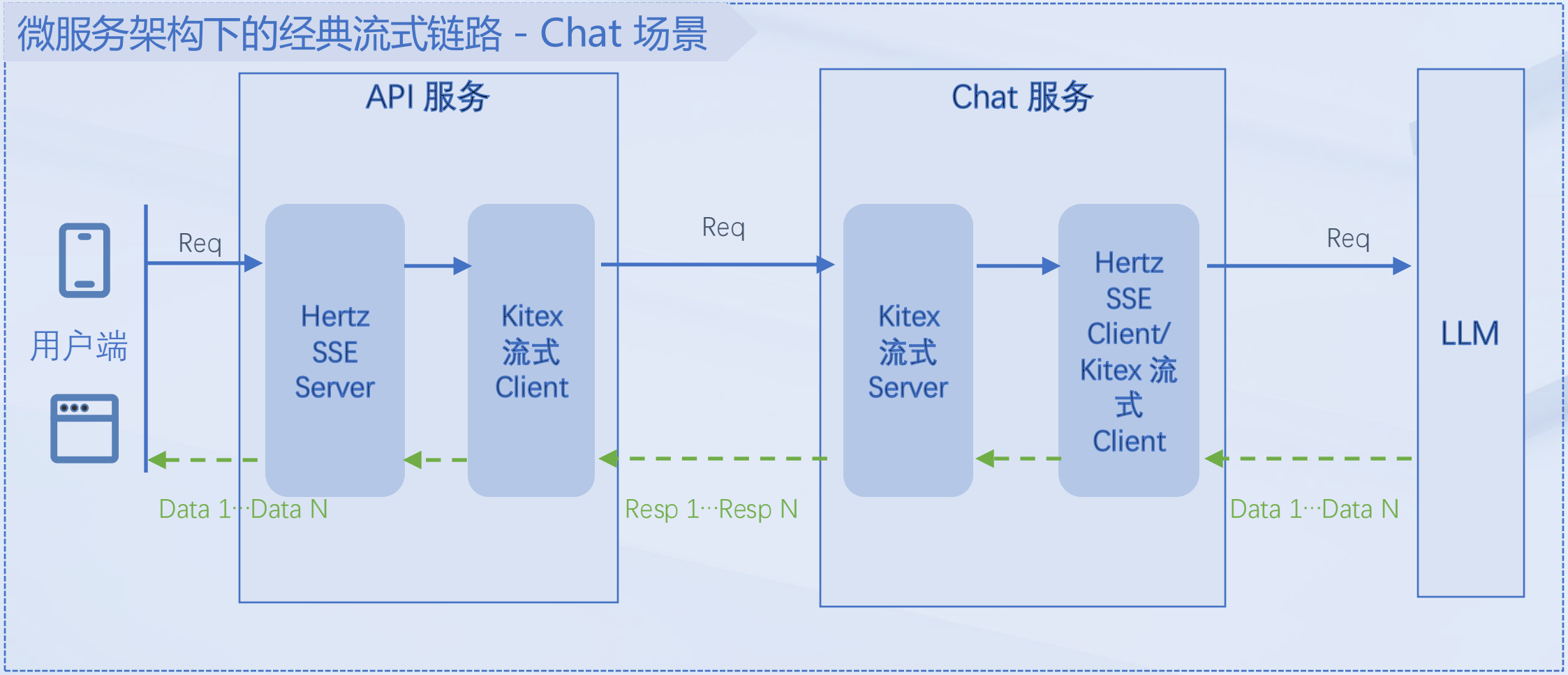
Hertz SSE - 端上交互



Kitex Streaming - 服务间交互
gRPC & TTHeader Streaming



大模型应用架构 | Chat 场景流式链路



流式工程实践问题

工程实践问题接踵而至

- 模型资源紧张，如何感知用户断开连接的信号，快速结束会话，节省资源？

- 用户反馈对话进行到一半报错，查看错误日志发现只有 context is canceled，是哪一环节出了问题？具体的错误原因又是什么？

- 如何对大模型应用进行有效监控？如何增强 Metrics 和 Trace 来适配流式场景？

02

流式工程实践增强

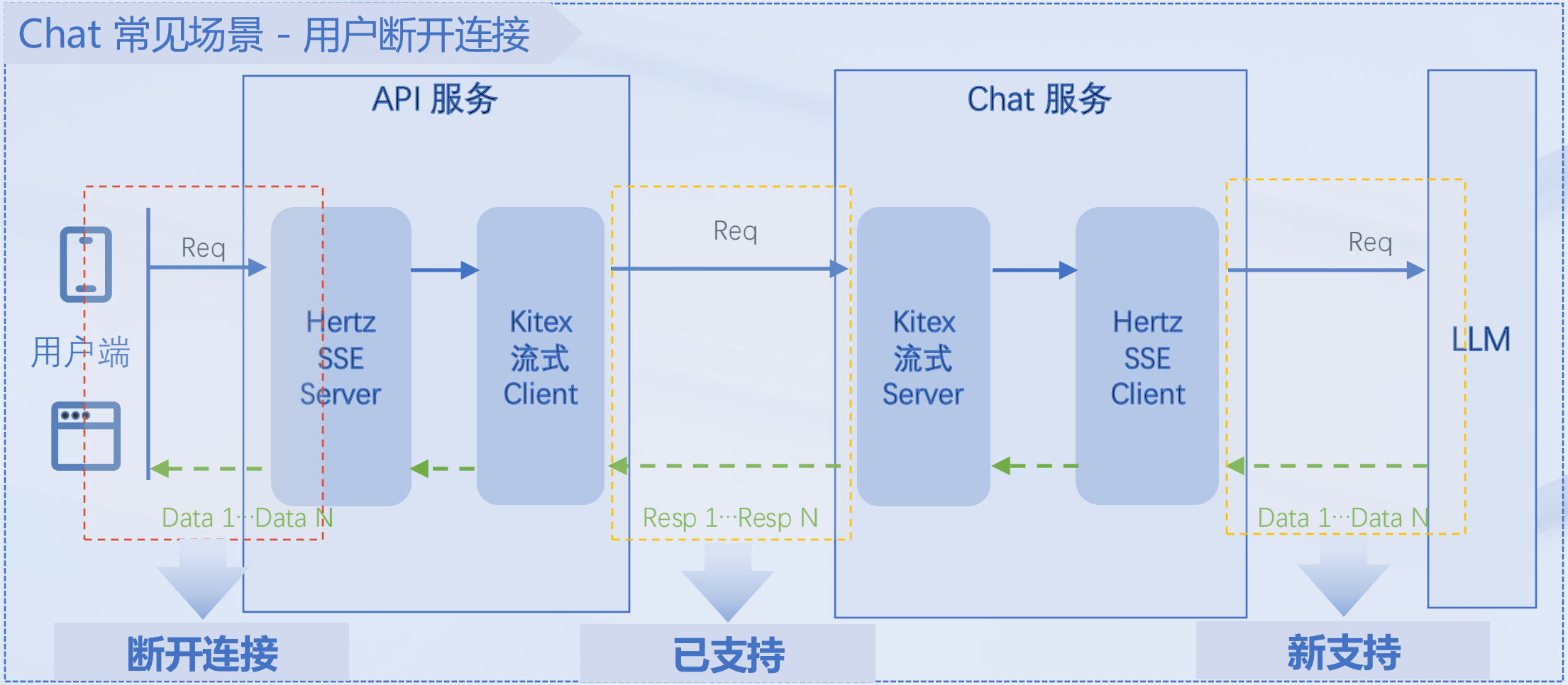
会话中断，流式异常，流式监控

会话中断 | 场景

会话中断场景五花八门

- 响应输出到一半，用户觉得 Prompt 不够准确，结束本轮会话后重新输入问题
- 响应输出时间过长，用户失去耐心
- 网络不稳定，连接中断
- 。。。。

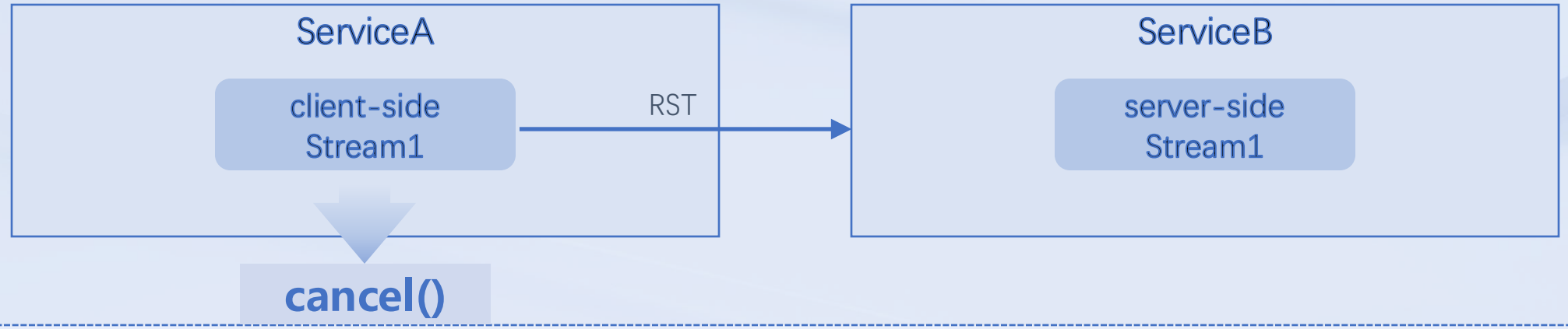
流式工程实践增强 | 主动结束流式调用，节省模型资源



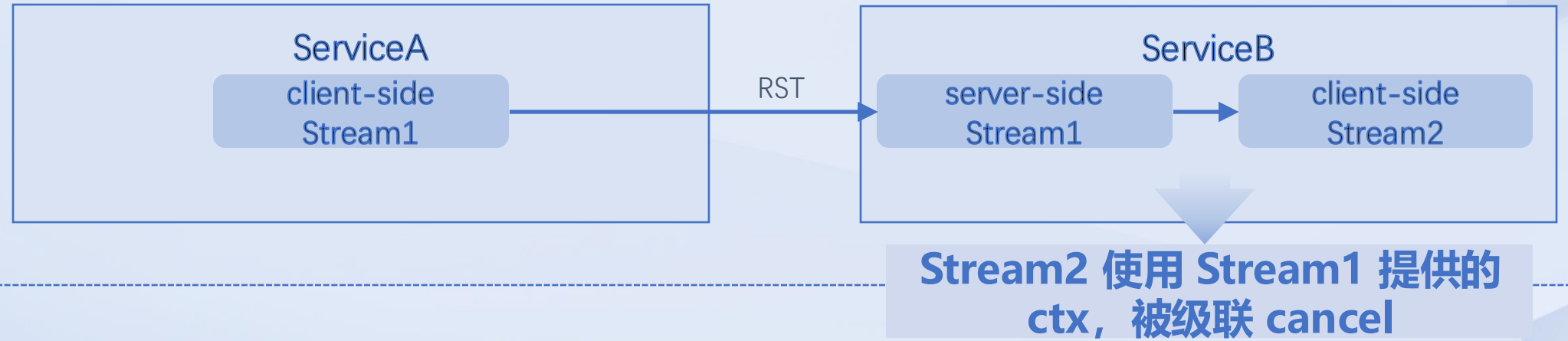
本质是一种上游主动结束流式调用的能力，控制 Stream 生命周期

流式工程实践增强 | 现状

Kitex gRPC 基于 ctx cancel 控制跨服务的 Stream 生命周期



Kitex gRPC 基于 ctx cancel 控制同服务的 Stream 生命周期



流式工程实践增强 | Hertz 原生支持 SSE, 补齐 ctx cancel 能力

Hertz 原生支持 SSE

```go
import (
    "github.com/cloudwego/hertz/pkg/app/client"
    "github.com/cloudwego/hertz/pkg/protocol/sse"
)


hzCli, err := client.NewClient()
hzCli.Do(ctx, req, resp)

hzCli.Do(ctx, req, resp)
reader, err := sse.NewReader(resp)
reader.ForEach(ctx, callback)
```

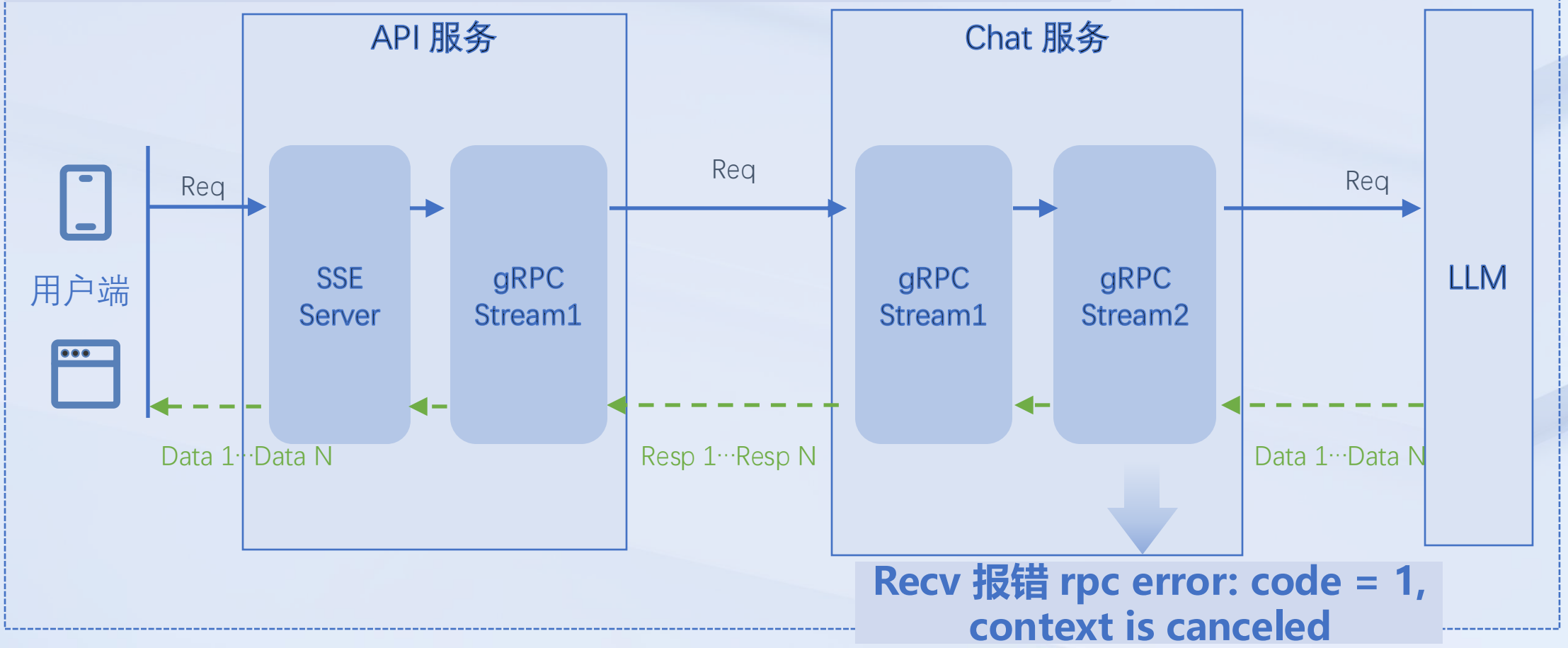
Hertz SSE Reader 补齐 ctx cancel 能力

```go
ctx, cancel := context.WithCancel(ctx)
err = hzCli.Do(ctx, req, resp)

reader, err = sse.NewReader(resp)
err = reader.ForEach(ctx, func(e *Event) error {
  if isBizSpecialFinishEvent(e) {
    cancel()
    return context.Canceled
  }
})
```

流式工程实践增强 | 错误描述优化

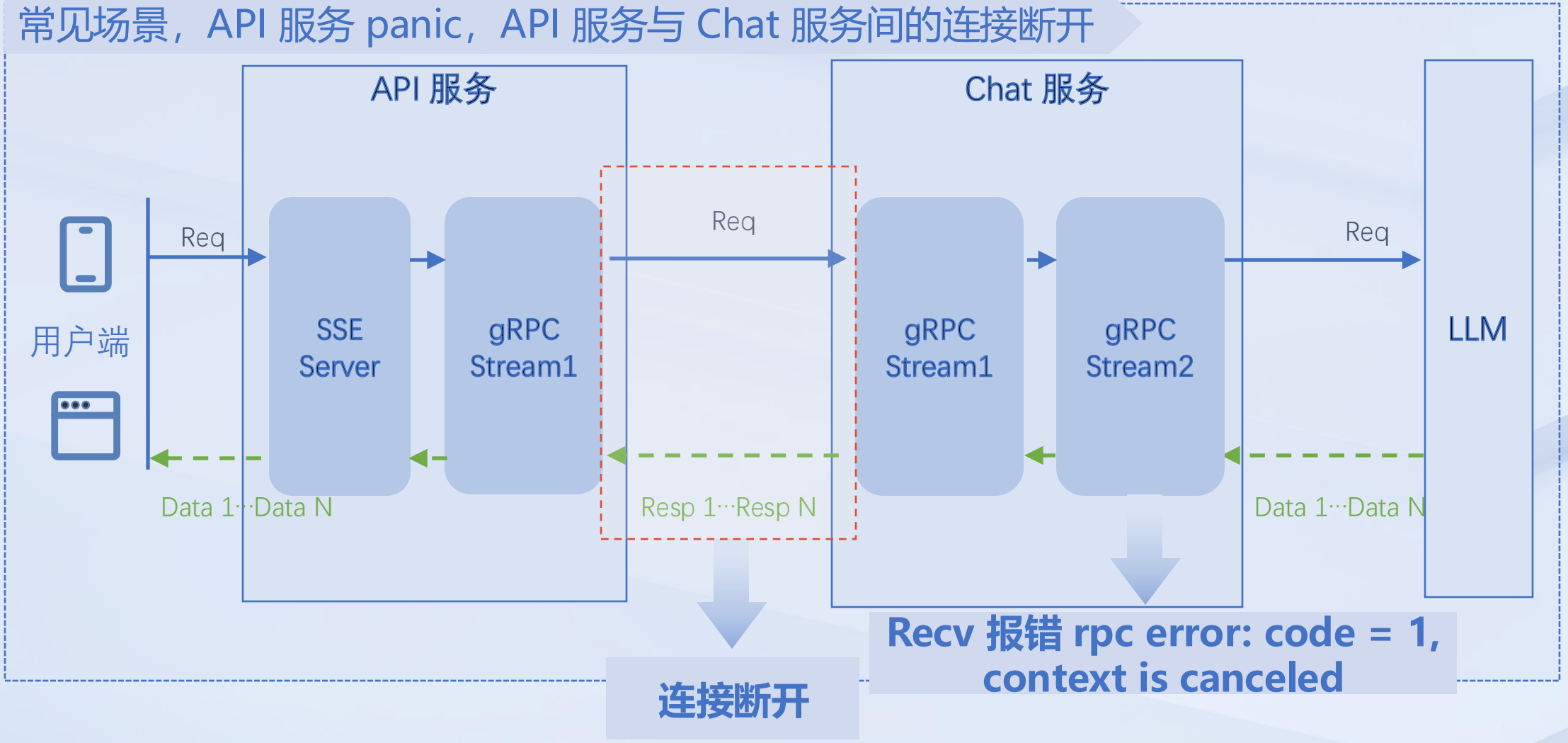
gRPC-go 常见错误：rpc error: code = 1, context is canceled



- 是 LLM 服务的问题么？
- 具体的错误原因是什么？

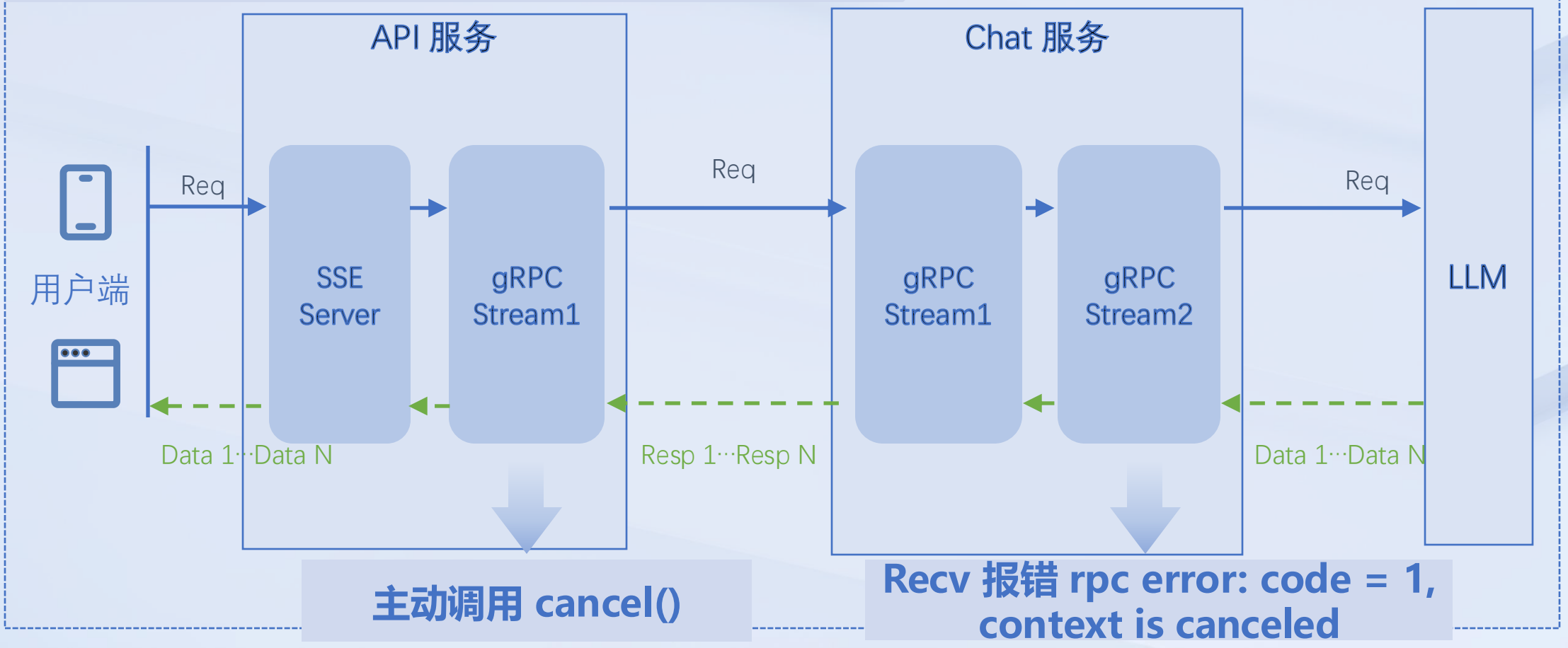
流式工程实践增强 | 错误描述优化

常见场景，API 服务 panic，API 服务与 Chat 服务间的连接断开



流式工程实践增强 | 错误描述优化

常见场景，API 服务主动调用 cancel() 结束流式调用



流式工程实践增强 | 错误描述优化

gRPC-go 错误描述的局限性

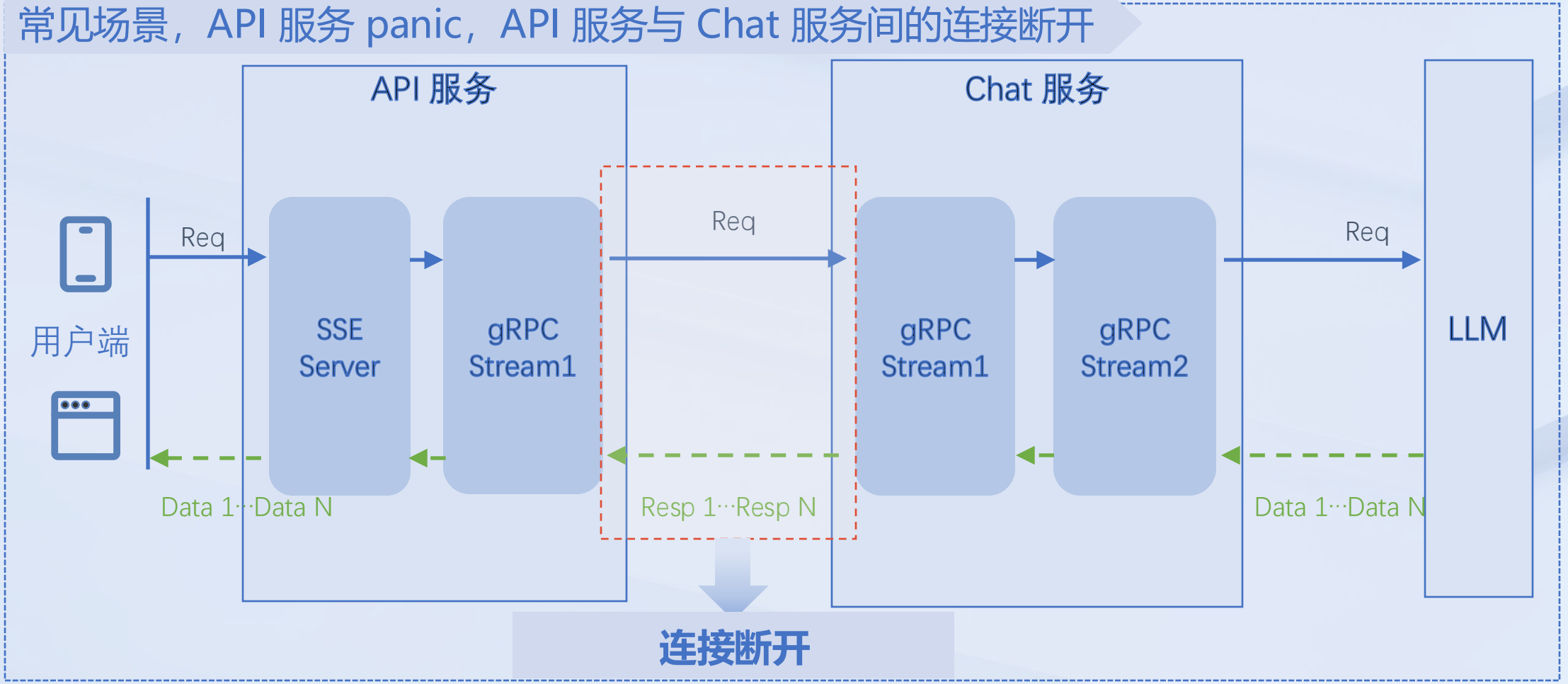
- 错误描述太过宽泛，无法具体对应错误场景
- 错误信息不足，无法快速定位到触发方

Kitex gRPC 错误描述优化，快速对应具体错误场景

- 错误分类为流级别错误和连接级别错误
- 错误描述精确对应具体错误场景
- 错误携带具体的触发方

流式工程实践增强 | 错误描述优化

常见场景，API 服务 panic，API 服务与 Chat 服务间的连接断开

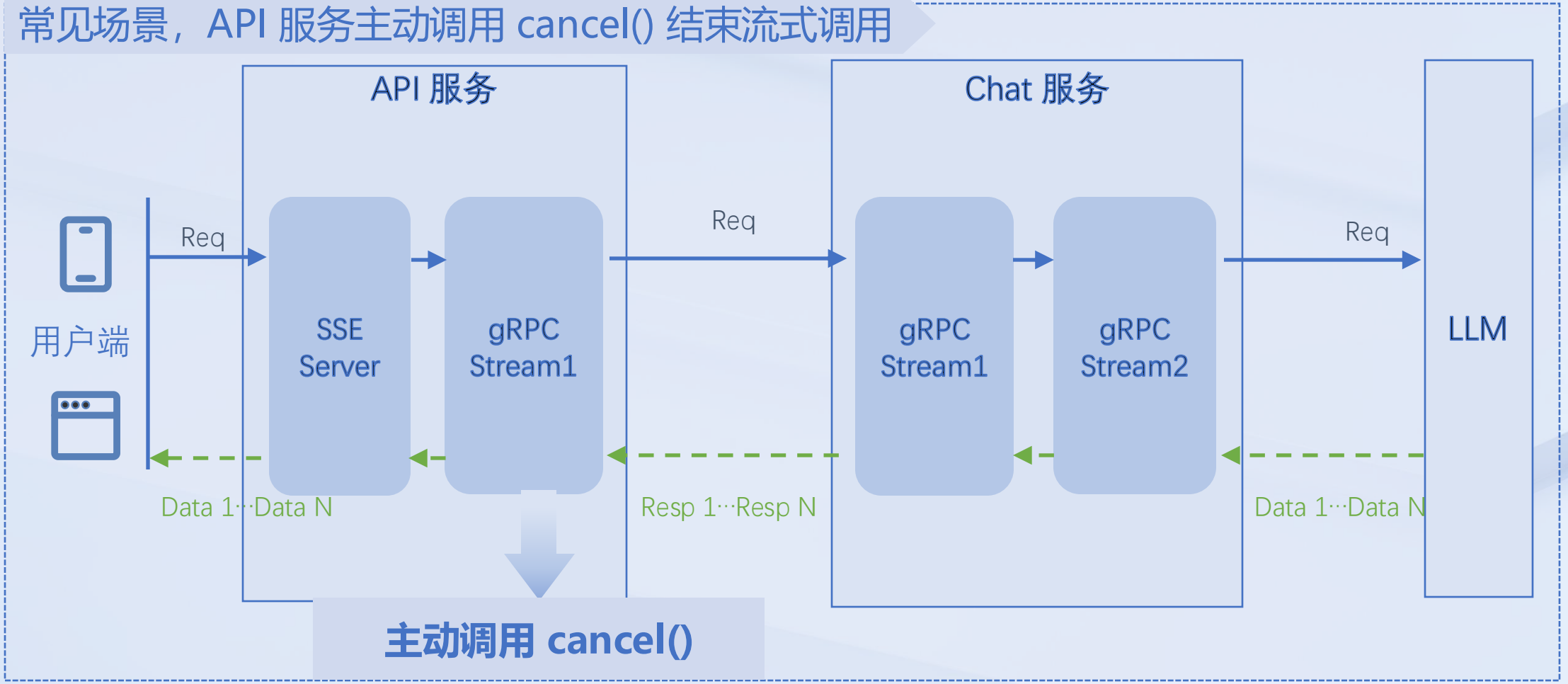


rpc error: code = 1, context is canceled

rpc error: code = 1, transport: connection EOF [triggered by API Service]

流式工程实践增强 | 错误描述优化

常见场景，API 服务主动调用 cancel() 结束流式调用



rpc error: code = 1, context is canceled

rpc error: code = 1, transport: RSTStream Frame received with error code: Canceled [triggered by API Service]

流式工程实践增强 | Metrics

传统 PingPong 监控指标在流式场景的“新”含义

- QPS
  - PingPong: 发送请求到接收响应
  - Streaming: 创建 Stream 到 Stream 生命周期结束
- Latency
  - PingPong: 发送请求到接收响应所经历的时间
  - Streaming: 创建 Stream 到 Stream 生命周期结束所经历的时间

可以将 PingPong 视作一个 Stream 顺序调用一次 Send 和 一次 Recv

含义并未发生本质改变

流式工程实践增强 | 引入 Recv/Send QPS

只关注 Stream QPS 和 Stream Latency 真的够么?

- 用户收到语句的间隔同样重要
- 同样是持续 3s 的会话，响应间隔不同，体验天差地别
  - 10 条响应以 300ms 的间隔均匀返回 
  - 10 条响应积压，最后 100ms 喷涌而出 

引入 Recv/Send QPS

- 基于 StreamEventReport 接口扩展
  - 每次 Recv/Send 调用，StreamEventReport 执行打点逻辑

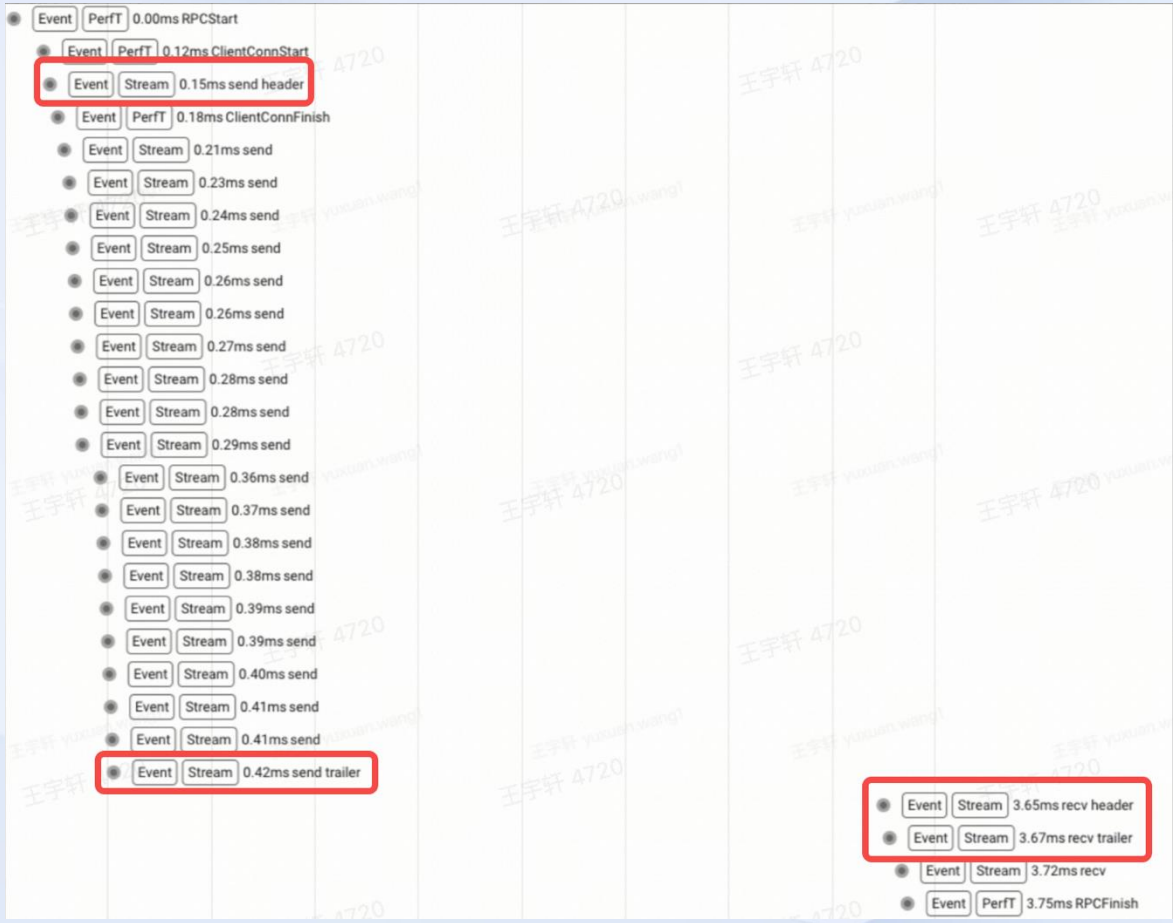
流式工程实践增强 | Trace

流式 Trace 增强，更好地把握 Stream 生命周期

- **新增改变 Stream 状态的关键事件**:
  - StreamSendHeader & StreamRecvHeader
  - StreamSendRst & StreamRecvRst
  - StreamSendTrailer & StreamRecvTrailer

Header/Rst/Trailer 能准确刻画 Stream 的状态流转，方便排查建流失败、非预期报错等疑难杂症

- **过滤 StreamSend & StreamRecv 事件**:
  一个 Stream 可能会持续很长时间，只保留前 N 条和后 N 条 StreamSend & StreamRecv 事件



流式能力深化建设

需要达成 可用 -> 好用 的飞跃

- 用户普遍反馈现有流式接口学习/使用成本高

- gRPC 错误描述优化难以应对复杂的级联 cancel 链路，排查成本高昂

03

流式能力 & 生态增强

StreamX 接口，自研流式协议 TTHeader Streaming，流式泛化

StreamX 接口提升流式易用性 | 背景

流式 handler 不暴露 ctx 参数，接口定义不统一
获取 ctx 必须使用 stream.Context()

issue

```
type Service interface {
    CallUnary(ctx context.Context, req *Request) (r *Response
    CallClient(stream Service_CallClientServer) (err error)
    CallServer(req *Request, stream Service_CallServerServer)
    CallBidi(stream Service_CallBidiServer) (err error)
}
```

```
func (s *ServiceImpl) CallServer(req *Request, stream
    ctx := stream.Context()
}
```

Send/Recv 接口无法传入 ctx 参数，扩展性差

issue

```
type Service_CallBidiServer interface {
    Recv() (*Request, error)
    Send(*Response) error
}
```

StreamX 接口提升流式易用性 | 背景

Option 配置作用域不明显

- 原有配置均能作用于 Ping-Pong 接口，流式接口是否生效？
  - client.WithRPCTimeout 无法作用于流式接口

流式中间件理解成本高，不符合使用直觉

- 原有中间件基于 Ping-Pong 接口设计，流式场景下 req/resp 参数语义完全不同

```go
func PingPongServerMiddleware(next endpoint.Endpoint) endpoint.Endpoint {
    return func(ctx context.Context, req, resp interface{}) (err error) {
        if arg, ok := req.(utils.KitexArgs); ok {
            // 处理 arg
        }
        err = next(ctx, req, resp)
        if res, ok := req.(*utils.KitexResult); ok {
            // 处理 res
        }
        return err
    }
}
```

接收请求 -> 处理 -> 返回响应

```go
func StreamingServerMiddleware(next endpoint.Endpoint) endpoint.Endpoint {
    return func(ctx context.Context, req, resp interface{}) (err error) {
        if stArg, ok := req.(*streaming.Args); ok {
            // 处理 stream
        }
        err = next(ctx, req, resp)
        // resp 始终为 nil
        return err
    }
}
```

接收 Stream -> 处理

StreamX 接口提升流式易用性 | 符合直觉的 ctx

流式 handler 暴露 ctx，接口定义统一

```
type Service interface {
    CallUnary(ctx context.Context, req *Request) (r *Response, err error)

    CallClient(ctx context.Context  stream Service_CallClientServer) (err error)

    CallServer(ctx context.Context  req *Request, stream Service_CallServerServer) (err error)

    CallBidi(ctx context.Context, stream Service_CallBidiServer) (err error)
}
```

Send/Recv 接口暴露 ctx 参数

```
type Service_CallBidiServer BidiStreamingServer[Request, Response]

type BidiStreamingServer[Req, Res any] interface {
    Recv(ctx context.Context) (*Req, error)

    Send(ctx context.Context, res *Res) error
}
```

StreamX 接口提升流式易用性 | Option 与中间件优化

Option 配置拆分，明确生效场景

```
cli, err := testservice.NewClient(targetService,
    client.WithUnaryOptions(
        client.WithUnaryRPCTimeout(timeout),
        client.WithUnaryMiddleware(unaryMW),
    ),

    client.WithStreamOptions(
        client.WithStreamMiddleware(stMW),
        client.WithStreamRecvMiddleware(recvMW),
        client.WithStreamSendMiddleware(sendMW),
    ),
)
```

原有配置依然生效，完全兼容

中间件拆分

```
func StreamingServerMiddleware(next sep.StreamEndpoint) sep.StreamEndpoint {
    return func(ctx context.Context, st streaming.ServerStream) (err error) {
        // 处理 stream
        return next(ctx, st)
    }
}
```

```
func StreamingServerRecvMiddleware(next sep.StreamRecvEndpoint) sep.StreamRecvEndpoint
    return func(ctx context.Context, stream streaming.ServerStream, message interface{
        err = next(ctx, stream, message)
        // 此时 message 已被反序列化好，处理 message
        return err
    }
}
```

```
func StreamingServerSendMiddleware(next sep.StreamSendEndpoint) sep.StreamSendEndpoint {
    return func(ctx context.Context, stream streaming.ServerStream, message interface{})
        // 处理 message
        return next(ctx, stream, message)
    }
}
```

自研协议 TTHeader Streaming | gRPC 痛点

gRPC 难以快速排查级联 cancel 链路问题

- 快速定位主动发起 cancel 的第一跳节点，并知晓 cancel 的具体原因，是解决问题的关键
- 和 Trace 一样，也需要一个能够携带链路信息的透传对象

gRPC 无法支持 context.WithCancelCause

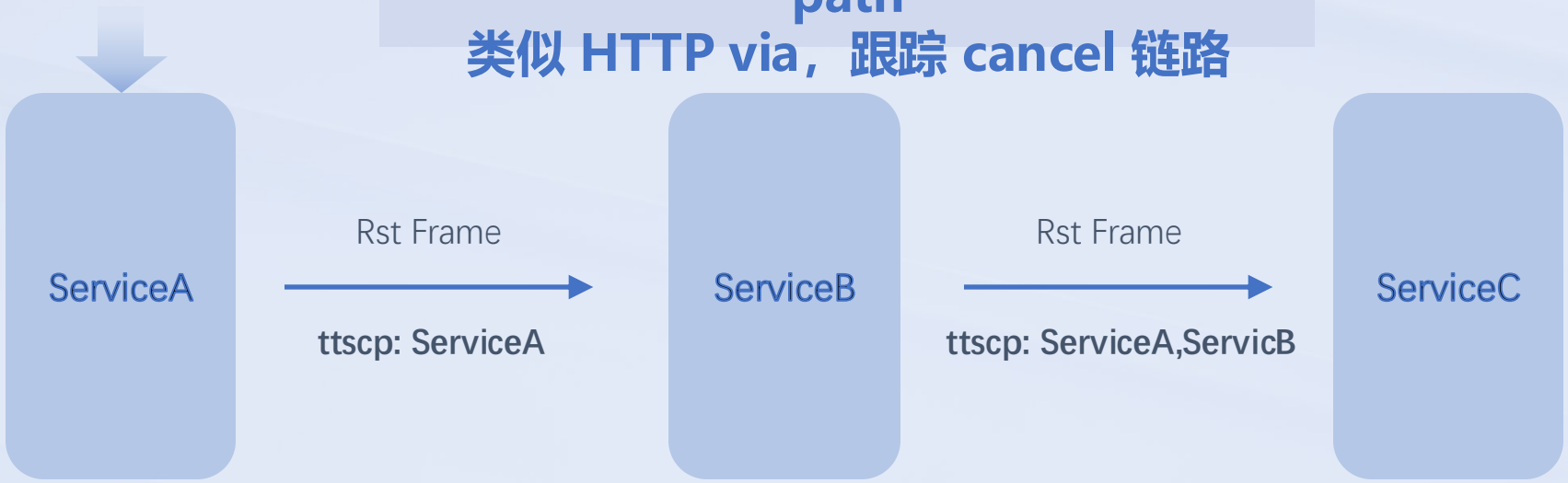
- cancel 上升为业务行为，存在传递自定义 cancel 异常的需求

gRPC 基于 HTTP2 RstStream Frame 实现 cancel，只支持传递 ErrCode，难以携带其它元信息

自研协议 TTHeader Streaming | 痛点优化

Rst Frame 携带链路元信息，高效应对级联 cancel

ttscp - ttheader streaming cancel path
类似 HTTP via，跟踪 cancel 链路



[ttstream error, code=12007]

[ttstream error, code=12007]
[canceled path: ServiceA]

[ttstream error, code=12007] **[canceled path: ServiceA -> ServiceB]**

自研协议 TTHeader Streaming | 痛点优化

支持 context.WithCancelCause，传递自定义 cancel 异常

- Rst Frame 携带 Payload，传递自定义 cancel 异常

```
ctx, cancel := context.WithCancelCause(ctx)
stream, err := kCli.CallServerStreaming(ctx, req)

for {
  resp, err := stream.Recv(ctx)
  if bizErr := processResp(resp); bizErr != nil {
    cancel(bizErr)
    return
  }
}
```

流式泛化 | 完备的生态支持

支持流式泛化 Client，一个泛化 Client 搞定流式/非流式场景

- 字节内部压测平台已广泛使用，测试流式接口

支持流式泛化 Server，支持处理 二进制/JSON/Map 数据

- 适配网关服务与 Mock 服务

kitexcall 支持流式调用，方便本地调试流式接口

- 体验类似 grpcurl，可从命令行与文件流式获取请求数据

04 总结 & 展望

总结

大模型应用架构概览

- Hertz SSE 与 Kitex Streaming 提供流式能力
- Chat 场景流式应用经典链路

流式工程实践增强

- 会话中断，加强生命周期控制
- 流式异常，加强问题定位能力
- 流式监控，加强流式消息观测能力

流式能力 & 生态增强

- StreamX 新接口，提高流式易用性
- 自研协议 TTHeader Streaming，高效排查级联 cancel 链路问题
- 流式泛化能力，为网关、测试服务提供完备生态支持

展望

最佳工程实践开源，反哺开源社区

- 沉淀字节内部丰富的流式接口使用和 Oncall 经验，形成流式使用指南并发布
- Metrics/Trace 流式增强提供 open-telemetry 版本

TTHeader Streaming 持续迭代

- 功能完全对齐 gRPC，支持基于协作的优雅退出
- 性能超越 Kitex gRPC

流式生态进一步增强

- WebSocket 增强，更好地适配语音场景

THANKS