语音实时化设计方案

针对实时ASR场景下WebSocket与HTTP灰度发布的矛盾，以及数据存储需求，以下是分步解决方案：

---

1. 核心架构分层设计

采用 **API Gateway + 实时WebSocket服务 + 异步存储管道** 的分层架构：

```
安卓设备 → API Gateway → WebSocket ASR服务 → 消息队列 → Java HTTP服务 → 数据库
```

---

2. 具体实施步骤

2.1 保留WebSocket实时接口

- **WebSocket服务独立化**：保持ASR核心服务基于WebSocket协议，确保100ms级实时性
- **协议优化**：使用Binary帧传输语音流，减少序列化开销（如Protobuf压缩数据）

2.2 灰度发布实现方案

方案一：API Gateway动态路由

```yaml
// Spring Cloud Gateway示例配置
spring:
  cloud:
    gateway:
      routes:
        - id: asr_websocket_route
          uri: lb://asr-service
          predicates:
            - Header=Gray-Release, false
          filters:
            - WebsocketRoutingFilter

        - id: asr_gray_route
```

```
          uri: lb://asr-gray-service
          predicates:
            - Header=Gray-Release, true
```

方案二：客户端SDK分流

```
// 安卓客户端代码示例
fun connectAsr(): WebSocketClient {
    val grayStatus = fetchGrayStatusFromHttp() // 先请求HTTP接口获取灰度状态
    return if (grayStatus) {
        WebSocketClient("wss://gray.asr.example.com")
    } else {
        WebSocketClient("wss://prod.asr.example.com")
    }
}
```

2.3 数据异步存储管道

```
# ASR服务处理完成后发送到Kafka
async def handle_asr_result(text):
    producer.send('asr-results-topic',
                  key=user_id,
                  value=json.dumps({
                    "text": text,
                    "metadata": {...}
                  }))

# Java服务消费并存储
@KafkaListener(topics = "asr-results-topic")
public void saveToDB(ConsumerRecord<String, String> record) {
    AsrResult result = parse(record.value());
    databaseRepository.save(result);
}
```

2.4 监控与熔断机制

```
# Istio VirtualService配置灰度流量
apiVersion: networking.istio.io/v1alpha3
```

```yaml
kind: VirtualService
metadata:
  name: asr-websocket
spec:
  hosts:
  - asr-service
  http:
  - match:
    - headers:
        x-gray-release:
          exact: "true"
    route:
    - destination:
        host: asr-gray-service
  - route:
    - destination:
        host: asr-prod-service
```

---

3. 关键优化点

1. **连接预热**：WebSocket长连接复用，减少TCP握手延迟
2. **二进制传输**：语音流采用Opus编码+Protobuf封装，降低带宽占用30%+
3. **分级降级**：网络抖动时自动切换语音压缩率（16kbps→8kbps）
4. **实时监控**：Prometheus+Grafana监控P99延迟、丢包率

---

4. 技术选型建议

组件	推荐方案	替代方案
API Gateway	Spring Cloud Gateway	Envoy, Nginx
服务发现	Nacos	Consul, Eureka
消息队列	Apache Kafka	RabbitMQ, Pulsar
流量治理	Istio	Linkerd

协议序列化	Protobuf	FlatBuffers

---

5. 实施路线图

1. Phase 1：搭建WebSocket服务与HTTP灰度控制端点（2周）
2. Phase 2：实现Kafka异步存储管道（1周）
3. Phase 3：集成Istio流量镜像做灰度验证（2周）
4. Phase 4：全链路压测与容灾演练（1周）

通过该方案，既保持了实时语音场景的WebSocket优势，又通过HTTP接口实现灵活的分流控制，异步存储机制确保数据可靠性。建议优先实施API Gateway方案，逐步引入服务网格增强治理能力。