C++音视频

对齐大厂音视频岗面试标准，我将 W1~W4 的技术栈、C++/QML 分工、具体用途、关键实现点拆解到底。原则只有一个：媒体管线 **100% C++**，**QML** 只做 **UI** 与状态展示。

核心分工原则（面试必讲）

层级	语言	职责	绝不越界
Media Engine	`C++`	拉流、解复用、解码、AVSync、JitterBuffer、硬解管理、OpenGL 渲染线程	不碰 `QQuickItem`，不依赖 Qt 事件循环
UI / Business	`QML`	播放控制、进度条、状态提示、调试面板、主题切换	不读 `AVFrame`，不调 `avcodec_*`，不阻塞主线程
Bridge	`C++ (QObject)`	`Q_PROPERTY` 暴露状态、`Q_INVOKABLE` 暴露控制接口、`QQuickFramebufferObject` 渲染视频纹理	仅传递数据指针/ID，不跨线程传递裸对象

W1~W4 逐周技术拆解

W1：FFmpeg 拉流 +解码 +OpenGL 渲染跑通

技术	语言	用途	关键实现点
`libavformat`	C++	打开文件/网络流，读取 `AVPacket`，解析音视频流信息	`avformat_open_input` → `avformat_find_stream_info` → `av_read_frame`
`libavcodec`	C++	软解 H.264/H.265 视频、AAC/Opus 音频到 `AVFrame`	`avcodec_send_packet` / `avcodec_receive_frame` 异步非阻塞调用
`libswscale`	C++	像素格式转换（YUV420P → RGBA）仅作为 **CPU** 降级方案	实际渲染应弃用，改由 OpenGL Shader 做 YUV→RGB

技术	语言	用途	关键实现点
`OpenGL 3.3+`	C++	GPU 渲染视频帧：YUV 纹理上传、Shader 转换、双缓冲	顶点着色器传 UV 坐标，片段着色器用 `texture2D` 采样 Y/U/V 平面，矩阵乘法转 RGB
`std::thread` / `std::mutex` / `std::condition_variable`	C++	解复用线程、解码线程、渲染线程隔离，线程安全队列	用 `std::queue<AVPacket*>` + 条件变量控制生产消费，避免内存碎片
`QML`	QML	仅显示“播放/暂停”按钮 + FPS 文本	通过 `Q_PROPERTY` 绑定 `fps`，不干预渲染逻辑

□ □□□□：1080p MP4 循环播放稳定 60 FPS，`valgrind`/`AddressSanitizer` 零泄漏，CPU 占用 `<15%`（软解）。

□□ W2：接入 SRT/UDP，实现基础 AVSync

技术	语言	用途	关键实现点
`FFmpeg SRT 协议支持` 或 `libsrt`	C++	替换文件 I/O 为低延迟网络传输	FFmpeg 原生支持 `srt://` 协议；或 `srt_create_socket` + `avio_alloc_context` 注入自定义读写回调
`SRTO_LATENCY` / `SRTO_TSBPDDELAY`	C++	控制 SRT 缓冲延迟，抗网络抖动	设置 `latency=120`（毫秒级），面试需解释 TSBPD（Time-Sender-Based Packet Delivery）原理
`音频主时钟`	C++	以音频播放进度为基准，计算音画时间差	`audio_clock = sample_pos / sample_rate`；`video_delay = video_pts - audio_clock`
`动态丢帧/重复帧`	C++	根据 `video_delay` 调整渲染节奏	`delay > 40ms` 丢帧；`delay < -30ms` 重复渲染上一帧；`std::this_thread::sleep_until` 精准睡眠
`QAudioSink` (Qt6) 或 `PortAudio`	C++	音频播放输出，提供高精度时钟源	回调函数中更新 `audio_clock`，确保时钟与硬件播放进度严格一致
`QML`	QML	网络状态指示、延迟显示、手动同步开关	绑定 `latency_ms`、`sync_status`，提供 UI 调试入口

□ □□□□：局域网 1v1 实时流，端到端延迟 ≤200ms，音画同步误差 ≤±20ms（手机慢动作录制验证）。

□□ W3: 硬解 Fallback + 零拷贝渲染 + 弱网 JitterBuffer

技术	语言	用途	关键实现点
`AVHWDeviceContext` / `AVHWFramesContext`	C++	管理硬件解码上下文（VAAPI/DXVA2/VideoToolbox）	`av_hwdevice_ctx_create` → 绑定到 `AVCodecContext` → 捕获 `AVERROR(EAGAIN)` 降级软解
`PBO (Pixel Buffer Object)` 或 `EGLImageKHR`	C++	零拷贝渲染: GPU 解码帧直接映射为 OpenGL 纹理	`glGenBuffers` + `glMapBufferRange` 避免 `glTexImage2D` CPU-GPU 同步阻塞; Linux 用 `EGLImage` 直通
动态 `Jitter Buffer`	C++	抗弱网: 根据 RTT/丢包率动态调整缓冲队列大小	滑动窗口计算 `jitter = max(inter_arrival) - min(inter_arrival)`; 丢包 >10% 扩容 +插值, <3% 缩容降延迟
`PLC (Packet Loss Concealment)`	C++	音频丢包隐藏（基础版: 线性插值/静音填充）	音频帧丢失时, 用上一帧末尾波形外推, 避免“咔嗒”爆音
`tc qdisc` (Linux) / `Clumsy` (Win)	工具	模拟弱网: 丢包、延迟、乱序、带宽限制	`tc qdisc add dev eth0 root netem loss 15% delay 50ms 20ms`
`perf` / `RenderDoc` / `nvidia-smi`	工具	性能剖析: CPU 热点、GPU 渲染管线、显存占用	`perf record -g ./app` → `perf report`; RenderDoc 抓帧验证零拷贝路径
`QML`	QML	实时性能面板: FPS、延迟、JitterBuffer 长度、硬解状态	使用 `ChartView` 或自定义 `Canvas` 绘制波形, 绑定 C++ 信号

□ □□□□: 压测报告（PDF/Markdown）含: 15% 丢包卡顿率 <2%、硬解 CPU <8%、PBO 渲染无 `glReadPixels`、72h 内存波动 <3%。

□□ W4: QML 对接 + 状态机 + 日志/Profiler 注入

技术	语言	用途	关键实现点
`QQuickFramebufferObject` 或 `QSGTexture`	C++	将 OpenGL 渲染上下文安全嵌入 QML 场景图	重写 `createRenderer()` 返回自定义 `Renderer`, 在 `render()` 中绑定 FBO 或纹理, 禁用 **`QQuickPaintedItem`**

技术	语言	用途	关键实现点
`QStateMachine` (Qt6)	C++	播放生命周期管理: `IDLE → CONNECTING → STREAMING → RECONNECTING → ERROR`	状态迁移绑定网络事件/用户操作，避免 `if-else` 嵌套地狱
`spdlog` 或 `QtLogging`	C++	结构化日志：时间戳、线程 ID、模块名、性能指标	`SPDLOG_INFO("decode", "pts={} delay={}ms", pts, delay)`；异步日志线程防阻塞
`CMake` + `vcpkg`/`Conan`	构建系统	跨平台依赖管理、编译选项优化（`-O3`、`-march=native`）	显式链接 `FFmpeg::avformat` 等 Modern CMake Target，避免硬编码路径
`QML`	QML	完整演示端：自定义按钮、主题切换、调试面板、状态反馈	使用你提供的 `CustomButton`，通过 `Qt.binding()` 响应引擎状态变化
`draw.io` / `PlantUML`	文档	架构图、线程模型、数据流图、状态迁移图	面试直接投屏讲解，标注线程边界与队列容量

□ □□□□：完整可运行 Demo + 架构图 + 压测报告 + `perf`/RenderDoc 截图 + 5 分钟演示视频。

技术栈归属速查表（面试防御用）

模块	**C++**	**QML**	说明
网络拉流 (SRT/UDP)	□	□	`avio` 回调或 `libsrt` 封装
解复用/解码 (FFmpeg)	□	□	纯 C++ 异步管线
音视频同步 (AVSync)	□	□	音频主时钟 + 丢帧/重复帧逻辑
硬解 Fallback	□	□	`AVHWDeviceContext` 管理
零拷贝渲染 (PBO/EGL)	□	□	OpenGL + `QQuickFramebufferObject`
Jitter Buffer / PLC	□	□	动态队列 + 音频插值
状态机/日志/性能探针	□	□	`QStateMachine` + `spdlog`
UI 控件/主题/动画	□	□	纯 QML，绑定 `Q_PROPERTY`
视频画面显示	⚠ C++ 渲染，QML 占位	□	`QQuickItem` 仅占位，纹理由 C++ 注入

面试怎么讲这套技术链？（话术模板）

> “项目采用**C++**媒体引擎**+ QML**表现层严格分离架构。底层基于FFmpeg实现解复用与软硬解码管线，通过`AVHWDeviceContext`管理硬件解码器，失败时自动Fallback到软解。渲染端使用PBO零拷贝技术将GPU帧直接映射为OpenGL纹理，通过`QQuickFramebufferObject`安全注入QML场景图，彻底避开CPU-GPU同步瓶颈。
>
> 音视频同步采用音频主时钟策略，根据PTS与硬件播放进度差值动态丢帧/重复帧，同步误差控制在±15ms。弱网层实现动态Jitter Buffer，基于RTT方差与丢包率滑动窗口调整缓冲深度，配合基础PLC保证15%丢包下卡顿率<2%。全链路通过`spdlog`埋点与`perf`/RenderDoc压测，首帧≤450ms，1080p@30fps硬解CPU <7%，内存72h压测波动<3%。”

本项目升级路线 阶段一 基础播放管线：脱离QtMultimedia，直接使用FFmpeg API实现解复用，软解码，音画同步，完成MP4与HLS基础播放 阶段二 硬件加速与渲染：接入DXVA2或NVDEC实现硬解，完成YUV到RGB色彩空间转换，使用OpenGL或DirectX实现零拷贝渲染，优化首屏耗时 阶段三 流控与质量保障：实现HLS自适应码率切换逻辑，设计多级缓冲策略，添加网络状态监控，实现卡顿统计与播放质量数据上报 阶段四 工程化架构重构：将播放引擎抽离为独立C加加动态库，提供线程安全接口，实现配置热加载，日志分级，内存池管理，补充单元测试与压测脚本

版本号 #2
由 叶子 创建于 13 四月 2026 15:50:23
由 叶子 更新于 18 四月 2026 15:28:17