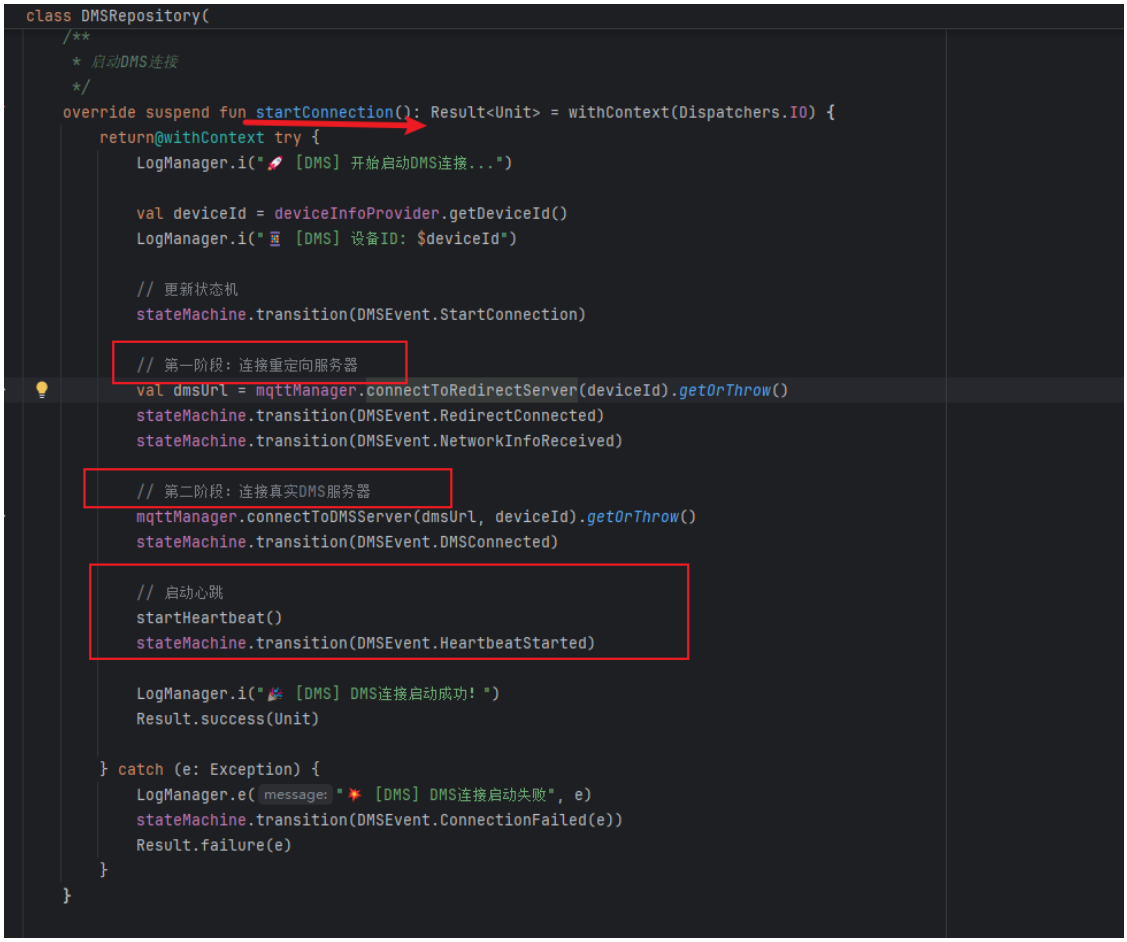
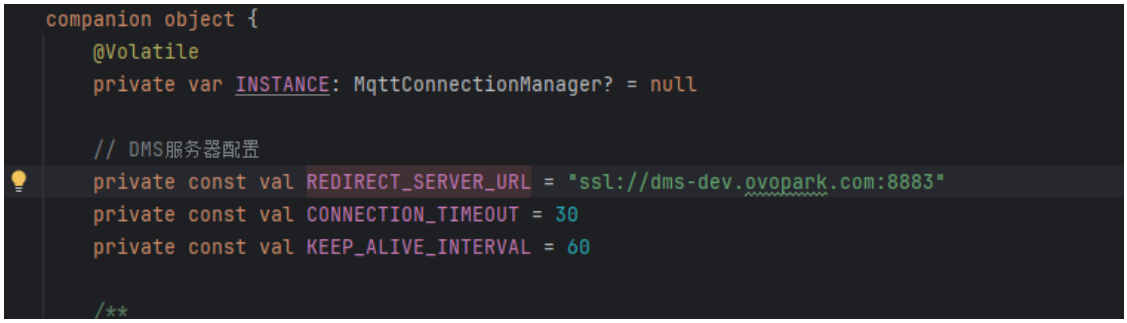
流程如下

整体流程

- 

```
class DMSRepository(
    /**
     * 启动DMS连接
     */
    override suspend fun startConnection(): Result<Unit> = withContext(Dispatchers.IO) {
        return@withContext try {
            LogManager.i("🚀 [DMS] 开始启动DMS连接...")

            val deviceId = deviceInfoProvider.getDeviceId()
            LogManager.i("📱 [DMS] 设备ID: $deviceId")

            // 更新状态机
            stateMachine.transition(DMSEvent.StartConnection)

            // 第一阶段：连接重定向服务器
            val dmsUrl = mqttManager.connectToRedirectServer(deviceId).getOrThrow()
            stateMachine.transition(DMSEvent.RedirectConnected)
            stateMachine.transition(DMSEvent.NetworkInfoReceived)

            // 第二阶段：连接真实DMS服务器
            mqttManager.connectToDMSServer(dmsUrl, deviceId).getOrThrow()
            stateMachine.transition(DMSEvent.DMSConnected)

            // 启动心跳
            startHeartbeat()
            stateMachine.transition(DMSEvent.HeartbeatStarted)

            LogManager.i("🎉 [DMS] DMS连接启动成功！")
            Result.success(Unit)

        } catch (e: Exception) {
            LogManager.e( message: "💥 [DMS] DMS连接启动失败", e)
            stateMachine.transition(DMSEvent.ConnectionFailed(e))
            Result.failure(e)
        }
    }
```

连接重定向服务器

- 设置连接选项
- 

```
    companion object {
        @Volatile
        private var INSTANCE: MqttConnectionManager? = null

        // DMS服务器配置
        private const val REDIRECT_SERVER_URL = "ssl://dms-dev.ovopark.com:8883"
        private const val CONNECTION_TIMEOUT = 30
        private const val KEEP_ALIVE_INTERVAL = 60

        /**
```

```
/**
 * 连接到重定向服务器（第一阶段）
 * @param deviceId 设备ID
 * @return 真实DMS服务器地址
 */
suspend fun connectToRedirectServer(deviceId: String): Result<String> = withContext(Dispatchers.IO) {
    LogManager.i("📞 [MQTT] 正在连接重定向服务器...")

    try {
        if (isConnecting.get()) {
            return@withContext Result.failure(Exception("正在连接中，请勿重复操作"))
        }

        isConnecting.set(true)
        _connectionStatus.tryEmit(ConnectionStatus.Connecting)

        // 创建MQTT客户端
        val client = MqttAndroidClient(context, REDIRECT_SERVER_URL, deviceId)
        mqttClient.set(client)

        // 设置连接选项
        val connectOptions = createConnectOptions(deviceId)

        // 设置回调
        client.setCallback(createRedirectMqttCallback())

        LogManager.i("🔑 [MQTT] 认证信息 - 用户名: $deviceId")
```

```
/**
 * 创建连接选项
 */
private fun createConnectOptions(deviceId: String): MqttConnectOptions {
    return MqttConnectOptions().apply {
        userName = deviceId
        password = generatePassword(deviceId).toCharArray()
```

```
/**
 * 生成认证密码
 */
private fun generatePassword(clientId: String): String {
    val timestamp = System.currentTimeMillis() / 1000 / 100 // 秒/100
    val raw = "ovopark_${clientId}_${timestamp}"
    return md5(raw)
}
```

- 执行连接

```
        // 执行连接
        val token = client.connect(connectOptions)
        token.waitForCompletion( timeout: CONNECTION_TIMEOUT * 1000L)

        if (!client.isConnected) {
            throw Exception("重定向服务器连接失败")
        }

        isConnected.set(true)
        _connectionStatus.tryEmit(ConnectionStatus.Connected)
        LogManager.i("✅ [MQTT] 重定向服务器连接成功")

        // 订阅入网主题
        subscribeToJoinNetworkTopics(deviceId)

        // 等待入网信息
        val realDMSUrl = waitForJoinNetworkMessage()

        LogManager.i("📍 [MQTT] 收到真实DMS服务器地址: $realDMSUrl")
        Result.success(realDMSUrl)
```

```kotlin
/**
 * 订阅入网相关主题
 */
private suspend fun subscribeToJoinNetworkTopics(deviceId: String) {
    val topics = arrayOf(
        "/elogger/#",
        "/elogger/$deviceId/verify_pwd",
        "/elogger/$deviceId/join_network",
        "/elogger/$deviceId/ctrlDevice"
    )
    val qos = intArrayOf(1, 1, 1, 1)

    subscribe(topics, qos)
    LogManager.i("👂 [MQTT] 开始监听入网信息...")

}
```

```kotlin
/**
 * 订阅主题
 * @param topics 主题数组
 * @param qos 服务质量数组
 */
suspend fun subscribe(topics: Array<String>, qos: IntArray): Result<Unit> = withContext(Dispatchers.IO) {
    try {
        val client = mqttClient.get() ?: throw Exception("MQTT客户端未初始化")

        if (!client.isConnected) {
            throw Exception("MQTT客户端未连接")
        }

        val token = client.subscribe(topics, qos)
        token.waitForCompletion( timeout: 10000)

        LogManager.i("👂 [MQTT] 主题订阅成功: ${topics.joinToString( separator: ", ")}")
        Result.success(Unit)
    } catch (e: Exception) {
        LogManager.e( message: "👂 [MQTT] 主题订阅失败", e)
        Result.failure(e)
    }
}

/**
```

连接真实DMS服务器

```kotlin
            subscribeToJoinNetworkTopics(deviceId)

            // 等待入网信息
            val realDMSUrl = waitForJoinNetworkMessage()

            LogManager.i("📍 [MQTT] 收到真实DMS服务器地址: $realDMSUrl")
            Result.success(realDMSUrl)

        } catch (e: Exception) {
            LogManager.e( message: "❌ [MQTT] 重定向服务器连接失败", e)
            _connectionStatus.tryEmit(ConnectionStatus.Failed(e))
            Result.failure(e)
        } finally {
            isConnecting.set(false)
        }
    }
```

```kotlin
    /**
     * 等待入网消息
     */
    private suspend fun waitForJoinNetworkMessage(): String = withContext(Dispatchers.IO) {
        LogManager.i("⏳ [MQTT] 开始等待入网消息（超时30秒）...")

        return@withContext withTimeoutOrNull( timeMillis: 30000) { // 30秒超时
            joinNetworkChannel.receive()
        } ?: run {
            LogManager.w("⏰ [MQTT] 等待入网消息超时，使用默认DMS服务器地址")
            // 超时后使用默认地址，让连接继续进行
            "ssl://dms2-test-001.ovopark.com:8883"
        }
    }

    /**
```

```kotlin
/**
 * 连接到DMS服务器（第二阶段）
 * @param dmsUrl DMS服务器地址
 * @param deviceId 设备ID
 */
suspend fun connectToDMSServer(dmsUrl: String, deviceId: String): Result<Unit> = withContext(Dispatchers.IO) {
    LogManager.i("🎯 [MQTT] 正在连接真实DMS服务器: $dmsUrl")

    try {
        // 断开重定向连接
        disconnectCurrent()

        // 连接真实DMS服务器
        val client = MqttAndroidClient(context, dmsUrl, deviceId)
        mqttClient.set(client)

        val connectOptions = createConnectOptions(deviceId)
        client.setCallback(createDMSMqttCallback())

        val token = client.connect(connectOptions)
        token.waitForCompletion( timeout: CONNECTION_TIMEOUT * 1000L)

        if (!client.isConnected) {
            throw Exception("DMS服务器连接失败")
        }

        isConnected.set(true)
        LogManager.i("✅ [MQTT] DMS服务器连接成功，可以开始心跳")

        Result.success(Unit)

    } catch (e: Exception) {
        LogManager.e( message: "❌ [MQTT] DMS服务器连接失败", e)
        Result.failure(e)
    }
```

```kotlin
    /**
     * 启动心跳定时任务
     */
    private fun startHeartbeat() {
        heartbeatJob?.cancel()
        heartbeatJob = repositoryScope.launch {
            while (isActive && stateMachine.isConnected()) {
                try {
                    sendHeartbeat()
                    delay( timeMillis: 30_000) // 30秒间隔
                } catch (e: Exception) {
                    LogManager.e( message: "💔 [DMS] 心跳任务异常", e)
                    if (e is CancellationException) {
                        break
                    }
                    // 心跳失败，等待重连
                    delay( timeMillis: 5_000) // 5秒后重试
                }
            }
        }

        LogManager.i("💓 [DMS] 心跳任务已启动")
    }
```

心跳

```kotlin
    /**
     * 发送心跳
     */
    override suspend fun sendHeartbeat(): Result<Unit> = withContext(Dispatchers.IO) {
        return@withContext try {
            // 获取最新设备状态
            val deviceStatus = deviceInfoProvider.getDeviceStatus()
            _deviceStatus.value = deviceStatus

            // 构建心跳数据
            val heartbeatData = deviceStatus.toJson()

            // 发送心跳
            mqttManager.sendHeartbeat(heartbeatData).getOrThrow()

            LogManager.d("💓 [DMS] 心跳发送成功")
            Result.success(Unit)

        } catch (e: Exception) {
            LogManager.e( message: "💔 [DMS] 心跳发送失败", e)
            stateMachine.transition(DMSEvent.HeartbeatFailed(e))
            Result.failure(e)
        }
    }
```

```kotlin
/**
 * 发送心跳消息
 * @param heartbeatData 心跳数据
 */
suspend fun sendHeartbeat(heartbeatData: String): Result<Unit> = withContext(Dispatchers.IO) {
    try {
        val topic = "/monitor/report/Sync_MainDeviceInfo"
        publish(topic, heartbeatData,  qos: 1,  retained: false)
        LogManager.i("💓 [MQTT] 心跳发送成功")
        Result.success(Unit)
    } catch (e: Exception) {
        LogManager.e( message: "💔 [MQTT] 心跳发送失败", e)
        Result.failure(e)
    }
}
```