Android 接入手册

OEM-SDK...emo.zip

1. 接入集团Maven仓库**或参考Demo直接引用Maven-Local**

```
repositories {
        maven { url 'http://mvn.midea.com/nexus/content/groups/public/' }
}
```

2. 引入SDK

```
//
implementation "com.midea.oem.iot:core:1.3.12.21"
//
implementation "com.midea.oem.iot.base:OEM-Account-SDK:1.3.12.21-SNAPSHOT"
//
implementation "com.midea.oem.iot.base:OEM-Device-SDK:1.3.12.21-SNAPSHOT"
//MQTT
implementation "com.midea.oem.iot.base:OEM-Message-SDK:1.3.12.21-SNAPSHOT"
//
implementation  "com.midea.iot.sdk:MSmartSDK:8.0.9"
```

3. SDK的初始化

```
OKHttpManager.init(
        enableDebugLog = true,
        appSecret = ", OEM",
        appKey = ", OEM",
        encryptKey = ", OEM",
        userAgents = ",Http user-agent",
        appId = "AppID, OEM",
        environment = "dev-, sit-, uat- ,",
        domain = "us.dollin.net"
)

OEMAccountManager.init(this)
OEMDeviceManager.init("M-SmartClientID", "M-SmartClientSecret")

//sdk
OEMDeviceManager.init(appid"AppID, OEM",appSecret = ", OEM")

//SDK
val config = MSConfig().apply {
            this.serverHost = "${HTTPS_SCHEME}${OKHttpManager.environment}${OKHttpManager.domain}"
//https://${OKHttpManager.environment}${OKHttpManager.domain}
            this.enableLog = true //
            this.clientId = OEMDeviceManager.clientId //clientid
            this.clientSecret = OEMDeviceManager.clientSecret //clientSecret
        }

        MSInterface.getInstance().initSDK(context, MSInterface.WorkMode.OVERSEAS_OEM, config)
```

4. 用户注册

```
// emailnull
// phoneAreacodeOEMAccountManager.getArea()phoneCode
// countryCodeOEMAccountManager.getArea()regionCode
// userFlag"0""1"
// mobilenull
// verifyIdReceiver()
// password RequestEncryptUtils.encryptPassword("xxxxxxxx")
// verifyIdOEMAccoutmanaget.authVerifyId()

RegisterRequest request = new RegisterRequest(email , phoneAreacode, countryCode , userFlag , mobile ,
verifyIdReceiver = account, privateVersion = "1.0", password , verifyId);

OEMAccountManager.register(request, new Callback<BaseResponse<UserInfo>>() {
    @Override
    public void onResponse(Call<BaseResponse<UserInfo>> call, Response<BaseResponse<UserInfo>> response)
{
        UserInfo userInfo = response.body().getData();
    }

    @Override
    public void onFailure(Call<BaseResponse<UserInfo>> call, Throwable t) {

    }
});
```

5. 用户登录

```
// account
// password RequestEncryptUtils.encryptPassword("xxxxxxxx")
// OEMAccountManager.getLoginUser()

LoginRequest loginRequest = new LoginRequest(account, password, "");
OEMAccountManager.login(loginRequest, new Callback<BaseResponse<UserInfo>>() {
@Override
public void onResponse(@NonNull Call<BaseResponse<UserInfo>> call, @NonNull
Response<BaseResponse<UserInfo>> response) {
    Log.d(DEMO_LOG_TAG, "login success:" + OEMAccountManager.getAccessToken());
}

@Override
public void onFailure(@NonNull Call<BaseResponse<UserInfo>> call, @NonNull Throwable t) {
    Log.d(DEMO_LOG_TAG, "login failure:" + t.getMessage());
}
});
```

6. 监听登录状态

```
// LifecycleOwnercrashnull
// Token

OEMAccountManager.observeLoginState(LifecycleOwner, loginEvent -> {
if (loginEvent instanceof LoginEvent.LoginSuccess) {
    Log.e(DEMO_LOG_TAG, "observeLoginState : LoginEvent.LoginSuccess");
} else if (loginEvent instanceof LoginEvent.LoginFailure) {
    Log.e(DEMO_LOG_TAG, "observeLoginState : LoginEvent.LoginFailure");
} else if (loginEvent instanceof LoginEvent.KickOut) {
    Log.e(DEMO_LOG_TAG, "observeLoginState : LoginEvent.KickOut");
} else if (loginEvent instanceof LoginEvent.TokenRefresh) {
    Log.e(DEMO_LOG_TAG, "observeLoginState : LoginEvent.TokenRefresh");
} else if (loginEvent instanceof LoginEvent.LoginOut) {
    Log.e(DEMO_LOG_TAG, "observeLoginState : LoginEvent.LoginOut");
}
});
```

7. 其他接口的调用

```
//JavaDocCall<BaseResponse<T>> execute()enqueue()/
Call<BaseResponse<RegionResponse>> call = OEMAccountManager.getArea();
//
Response<BaseResponse<RegionResponse>> areaResponse = call.execute();
//
call.enqueue(new Callback<BaseResponse<RegionResponse>>() {
        @Override
        public void onResponse(Call<BaseResponse<RegionResponse>> call,
Response<BaseResponse<RegionResponse>> response) {
                //
        }

        @Override
        public void onFailure(Call<BaseResponse<RegionResponse>> call, Throwable t) {
                //
        }
});
```

8. 设备SDK接口调用示例

```
//
val data=fun bindDevice(request: BindDeviceRequest) = getMsNetApi().bindDevice(request)
//
 @POST("/midea/open/business/v1/ads")
    fun getAdsInfo(): Call<BaseResponse<AdsInfoResponse>>

    @POST("/midea/open/business/v1/token")
    fun refreshToken(@Body refreshTokenRequest: RefreshTokenRequest):
Call<BaseResponse<RefreshTokenResponse>>

    @POST("/midea/open/business/v1/bind")
    fun bindDevice(@Body bindDeviceRequest: BindDeviceRequest): Call<BaseResponse<Unit>>

    @POST("/midea/open/business/v1/unbind")
    fun unbindDevice(@Body unbindDeviceRequest: UnbindDeviceRequest): Call<BaseResponse<Unit>>

    @POST("/midea/open/business/v1/country/channel/retrieve")
    fun getCountryChannel(@Body request: GetCountryChannelRequest):
Call<BaseResponse<Array<GetCountryChannelResponse>>>

    /**  */
    @POST("/midea/open/business/v1/appliance/list")
    fun getDeviceList(): Call<BaseResponse<List<DeviceVO>>>

    /**  */
    @POST("/midea/open/sdk/v2/appliance/verification")
    fun verifyDevice(@Body verifyDeviceRequest: VerifyDeviceRequest):
Call<BaseResponse<VerifyDeviceResponse>>

    /**  */
    @POST("/midea/open/business/v1/product-catalogs")
    fun getProductCatalogs(): Call<BaseResponse<List<ProductCatalogsResponse>>>

    /**  */
    @POST("/midea/open/business/v1/products")
    fun getProducts(@Body getProductListRequest: GetProductListRequest):
Call<BaseResponse<List<GetProductsResponse>>>

    /**  */
    @POST("/midea/open/business/v1/guide")
    fun getGuide(@Body netConfigStepsRequest: NetConfigStepsRequest):
Call<BaseResponse<List<NetConfigSteps>>>

    /**  */
    @POST("/midea/open/business/v1/appliance/auth")
    fun authDevice(@Body netConfigAuthDeviceRequest: NetConfigAuthDeviceRequest):
Call<BaseResponse<Unit>>
```

```
    /** */
    @POST("/midea/open/business/v1/appliance/auth/status")
    fun getAuthStatus(@Body request: NetConfigAuthStatusRequest):
Call<BaseResponse<NetConfigAuthStatusResponse>>

    /***/
    @POST("/midea/open/business/v1/appliance/status")
    fun getDeviceStatus(@Body request: DeviceStatusRequest): Call<BaseResponse<DeviceStatusResponse>>

    /**  json */
    @POST("/midea/open/business/v1/appliance/control")
    fun control(@Body request: JsonControlDeviceRequest): Call<BaseResponse<ControlDeviceResponse>>

    /** oemjson */
    @POST("/midea/open/business/v1/appliance/control/json")
    fun jsonControl(@Body request: OemJsonControlDeviceRequest): Call<BaseResponse<Unit>>

    /** 16 */
    @POST("/midea/open/business/v1/appliance/control/hexadecimal")
    fun control(@Body request: HexControlDeviceRequest): Call<BaseResponse<String>>

    /** oem */
    @POST("/midea/open/business/v1/appliance/fullQuery")
    fun queryFull(@Body request: FullQueryRequest):Call<BaseResponse<FullQueryResponse>>

    /**  */
    @POST("/midea/open/business/v1/device/command/{thingCode}")
    fun msCommand(@Path("thingCode") thingCode: String, @Body commands: RequestBody):
Call<BaseResponse<Unit>>

    /**  */
    @POST("/midea/open/business/v1/appliance/subscription")
    fun subscribe(@Body request: DeviceSubscriptionRequest):
Call<BaseResponse<DeviceSubscriptionResponse>>

    /**  */
    @POST("/midea/open/v1/appliance/subscription")
    fun unSubscribe(@Body request: DeviceUnSubscribeRequest): Call<BaseResponse<Unit>>

    /**  */
    @POST("/midea/open/v1/plugins/getPlugin")
    fun getDevicePlugin(@Body request: GetPluginRequest): Call<BaseResponse<GetPluginResponse>>

    /**  */
    @POST("/midea/open/business/v1/appliance/updateDeviceInfo")
    fun updateDeviceInfo(@Body request: UpdateDeviceInfoRequest): Call<BaseResponse<Unit>>

    /**  */
    @POST("/midea/open/business/v1/searchProducts")
    fun searchProducts(@Body request: SearchProductsRequest): Call<BaseResponse<SearchProductsResponse>>

    /**  */
    @POST("/midea/open/business/v1/appliance/share")
    fun shareDevice(@Body request: ShareDeviceRequest): Call<BaseResponse<Unit>>

    /**  */
    @POST("/midea/open/business/v1/appliance/share/receive")
    fun shareAccept(@Body request: ReceiveShareDeviceRequest): Call<BaseResponse<Unit>>

    @POST("/midea/open/business/v1/appliance/asyncQueryB5")
    fun queryB5(): Call<BaseResponse<Unit>>

    @POST("/midea/open/business/v1/appliance/info")
    fun getDeviceInfo(@Body request: GetDeviceInfoRequest): Call<BaseResponse<GetDeviceInfoResponse>>

    @POST("/midea/open/business/v1/area/city")
    fun getRegionId(@Body request: GetRegionIdRequest): Call<BaseResponse<GetRegionIdResponse>>


    @POST("/midea/open/business/v1/app2base/data/transmit")
    fun postTransmit(@Query("serviceUrl")serviceUrl:String, @Body request: GetTransmitRequest):
Call<BaseResponse<ResponseBody>>
```

9. 配网SDK调用示例：

```
/**
     * MS Token
     */
    fun initMSToken() {
        if (OEMDeviceManager.clientId.isEmpty() || OEMDeviceManager.clientSecret.isEmpty()) {
            logD(TAG, "")
            return
        }
        OEMDeviceManager.refreshToken(OKHttpManager.accessToken)
        MSInterface.getInstance().setAccessToken(OKHttpManager.accessToken)
        logD(TAG, " Ads portiddomain")
        getAdsInfo()
    }

    /** ADS */
    private val retryCount: AtomicInteger = AtomicInteger(30)

    /** ADS */
    private fun getAdsInfo() {
        OEMDeviceManager.getAdsInfo(object : Callback<BaseResponse<AdsInfoResponse>> {
            override fun onResponse(call: Call<BaseResponse<AdsInfoResponse>>, response:
Response<BaseResponse<AdsInfoResponse>>) {
                logD(TAG, " Ads ")
                getWifiChannel()
            }

            override fun onFailure(call: Call<BaseResponse<AdsInfoResponse>>, t: Throwable) {
                logD(TAG, " Ads , cause: ${t.message}")
                if (retryCount.get() > 0 && (t is ServerApiException && (t.code != "14005"))) {
                    Handler(Looper.getMainLooper()).postDelayed({ getAdsInfo() }, 2000L)
                }
                retryCount.decrementAndGet()
            }
        })
    }

    /**
     *
     */
    private fun getWifiChannel() {
        logD(TAG, "")
        //
        OEMDeviceManager.getChannel(GetCountryChannelRequest(regionService?.getSelectRegion()?.
regionCode ?: "")).enqueue(
            object : Callback<BaseResponse<Array<GetCountryChannelResponse>>> {
                override fun onResponse(call: Call<BaseResponse<Array<GetCountryChannelResponse>>>,
response: Response<BaseResponse<Array<GetCountryChannelResponse>>>) {
                    kotlin.runCatching {
                        logD(TAG, "")
                        if ((response.body()?.data?.size ?: 0) > 0) {
                            channelList.clear()
                        }
                        response.body()?.data?.forEach { item ->
                            val channel = MSCountryChannel(item.channelNum.toInt().toByte(), item.
channelCounts.toInt().toByte(), item.maxTransferPower.toInt().toByte(), item.dfs == "1")
                            logD(TAG, "\t $channel")
                            channelList.add(channel)
                        }
                    }
                }

                override fun onFailure(call: Call<BaseResponse<Array<GetCountryChannelResponse>>>, t:
Throwable) {
                    logD(TAG, " cause${t.message}")
                }
            }
        )
    }

    /**
     *
     */
    fun startBLEConfig(
        context: Context,
        bleConfigParams: BleConfigParams,
        callback: OEMConfigCallback
    ) {
        stopConfig()
        BLEConfigData.currentConfigBleDevice = null
        logD(TAG, "======> ")
        configJob = bleConfigParams.oemDevice.findStrategy().startConfig(context, bleConfigParams,
callback)
    }

    /**
     * AP
     */
    fun startAPConfig(
        context: Context,
        apConfigParams: APConfigParams,
        callback: OEMConfigCallback
    ) {
        stopConfig()
        BLEConfigData.currentConfigBleDevice = null
        logD(TAG, "======> AP")
        configJob = apConfigParams.findAPStrategy().startConfig(context, apConfigParams, callback)
    }

    @JvmStatic
    fun stopConfig() {
        logD(TAG, "<<>>")
        MSDeviceConfigManager.getInstance().stopConfigureDevice()
        BLEConfigData.currentConfigBleDevice?.let {
            BleManager.getInstance().disconnect(it)
        }
        if (::configJob.isInitialized && configJob.isActive) {
            configJob.cancel()
        }
    }

    /**
     *
     *
     */
    suspend fun updateDeviceName(thingCode: String, thingName: String): BaseResponse<Unit> {
        logD(TAG, ": $thingCode -> $thingName")
        return modifyApplianceInfo(thingCode, thingName)
    }

    /**
     *
     *
     */
    suspend fun updateConfigMessage(): BaseResponse<Unit> {
        logD(TAG, "")
        return updateDevice()
    }

    private lateinit var configJob: Job

    @Suppress("FunctionName")
    fun _debug_data_available(): Boolean {
        return channelList.isNotEmpty()
    }
```