CUGE Android SDK 开发者文档

注意，开发者在使用以下文档前，需先在CUGE管理后台申请建立业务线和应用ID。

目前CUGE Android SDK的最新版本为1.1.0，本篇文章对此SDK的用法进行详细说明，具体用法也可参考对应的Demo实现：

https://github.com/CUGE-Dev/cuge-android-demo

注意，从Android9.0系统开始，应用程序默认只允许使用HTTPS类型的网络服务，HTTP类型的网络请求因为有安全隐患默认不再被支持，受限于资金预算，我们搭建的CUGE SDK服务器使用的是HTTP，故在此补充说明为了能让程序使用HTTP所需要的配置。

（1）在res目录下新建一个xml文件夹，然后再在xml文件夹下新建一个network_config.xml文件，然后修改文件为如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true">
        <trust-anchors>
            <certificates src ="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

（2）修改AndroidManifest.xml来启用步骤1中创建的配置文件，在块里加入一行代码：android:networkSecurityConfig="@xml/network_config

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:networkSecurityConfig="@xml/network_config">
        ...
    </application>
```

一、项目配置

1. 在project下的build.gradle里引入如下三行依赖：

   maven { url 'http://47.107.119.93:8090/repository/maven-snapshots/'}
   maven { url 'http://47.107.119.93:8090/repository/maven-public/' }
   maven { url "https://jitpack.io" }

   ```
   buildscript {
       repositories {
           google()
   ```

```
        jcenter()
        maven { url 'http://47.107.119.93:8090/repository/maven-snapshots/'}
        maven { url 'http://47.107.119.93:8090/repository/maven-public/' }
        maven { url "https://jitpack.io" }
    }
}
allprojects {
    repositories {
        google()
        jcenter()
        maven { url 'http://47.107.119.93:8090/repository/maven-snapshots/'}
        maven { url 'http://47.107.119.93:8090/repository/maven-public/' }
        maven { url "https://jitpack.io" }
    }
}
```

2. 在 app 模块里的build.gradle里引入如下依赖：

```
implementation 'com.zzm.android_basic_library:cuge_android_sdk:1.1.0'
```

3. 在app模块下的res下新建raw文件夹，将在CUGE管理后台申请得到的分装好公钥的图片放入此文件夹下。
4. SDK初始化，在自定义的application的onCreate()中调用CUGEAndroidSDK.init()方法，onTerminate()中调用CUGEAndroidSDK.destroy()方法，具体用法如下代码所示：

```
class MyApplication : Application() {

    companion object{
       val scope = MainScope()
    }

    override fun onCreate() {
        super.onCreate()
        scope.launch(Dispatchers.Main.immediate) {
            CUGEAndroidSDK.init(
                applicationContext, 1,
                CryptoUtil.loadPubKeyFromImage(applicationContext,
R.raw.test_bizline), "test_bizline"
            )
        }
    }

    override fun onTerminate() {
        super.onTerminate()
        CUGEAndroidSDK.destroy()
        scope.cancel()
    }
}
```

其中 CUGEAndroidSDK.init()的函数原型如下：

```
/**
 * @param appContext application context
 * @param appId CUGE管理后台拿到的应用ID
 * @param publicKey CUGE管理后台给到的公钥
 * @param xBizLine CUGE管理后台所注册的业务线名称
 */
fun init(context: Context, appId: Int,  publicKey: ByteArray,  xBizLine:
String)
```

二、接口说明（SDK中网络请求用协程和Retrofit实现，故调用接口函数时需要在协程作用域里调用）

1. 登录注册

（1）发【手机】验证码（注册需要用到）

- 使用方式：

```
val result =CUGEAndroidSDK.authentication().sendMobileVerifyCode(mobile:
String)
```

- 参数说明：String类型的手机号码。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示验证码发送成功；result值为-1表示验证码发送失败。

（2）发【邮箱】验证码（注册需要用到）

- 使用方式：

```
val result =CUGEAndroidSDK.authentication().sendEmailVerifyCode(email:
String)
```

- 参数说明：String类型的邮箱账号。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示验证码发送成功；result值为-1表示验证码发送失败。

（3）【手机】注册

- 使用方式：

```
val mobileRegisterReqBody = MobileRegisterReqBody(mobile: String, password:
String, verifyCode: String)
val result
=CUGEAndroidSDK.authentication().mobileRegister(mobileRegisterReqBody)
```

- 参数说明：先用String类型的手机号、String类型的密码、String类型的验证码构建MobileRegisterReqBody对象，之后将MobileRegisterReqBody对象作为参数传进mobileRegister方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示手机号注册成功；result值为-1表示手机号注册失败。

（4）【邮箱】注册

- 使用方式：

```
val emailRegisterReqBody = EmailRegisterReqBody(email: String, password:
String, verifyCode: String)
val result
=CUGEAndroidSDK.authentication().emailRegister(emailRegisterReqBody)
```

- 参数说明：先用String类型的邮箱账号、String类型的密码、String类型的验证码构建EmailRegisterReqBody对象，之后将EmailRegisterReqBody对象作为参数传进emailRegister方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示邮箱注册成功；result值为-1表示邮箱注册失败。

(5) 【手机】登录

- 使用方式：

```
val mobileLoginReqBody = MobileLoginReqBody(mobile: String, password:
String)
val result = CUGEAndroidSDK.authentication().mobileLogin(mobileLoginReqBody)
```

- 参数说明：先用String类型的手机号、String类型的密码构建MobileLoginReqBody对象，之后将MobileLoginReqBody对象作为参数传进mobileLogin方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示手机登录成功；result值为-1表示手机登录失败。

(6) 【邮箱】登录

- 使用方式：

```
val emailLoginReqBody = EmailLoginReqBody(email: String, password: String)
val result  = CUGEAndroidSDK.authentication().emailLogin(emailLoginReqBody)
```

- 参数说明：先用String类型的邮箱账号、String类型的密码构建EmailLoginReqBody对象，之后将EmailLoginReqBody对象作为参数传进emailLogin方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示邮箱登录成功；result值为-1表示邮箱登录失败。

(7) 【qq】登录（此接口详情请见“三、可插拔的QQ登录模块”）

(8) 退出登录

- 使用方式：

```
  val result  = CUGEAndroidSDK.authentication().logout()
```

- 返回值说明：该函数返回一个Int类型的值，result值为0表示退出登录成功；result值为-1表示退出登录失败。

(9) 【手机】已登录用户更新密码

- 使用方式：

```
 val result  =
CUGEAndroidSDK.authentication().mobileUpdatePassword(mobilePassword: String)
```

- 参数说明：String类型的密码
- 返回值说明：该函数返回一个Int类型的值，result值为0表示手机号更新密码成功；result值为-1表示手机号更新密码失败；result值为2表示无法获取当前登录用户的UserID，可能是

SDK Token没有登录态；result值为3表示当前登陆账号没有绑定手机号，无法更新密码。

(10) 【邮箱】已登录用户更新密码

- 使用方式：

```
val result  =
CUGEAndroidSDK.authentication().emailUpdatePassword(emailPassword: String)
```

- 参数说明：String类型的密码
- 返回值说明：该函数返回一个Int类型的值，result值为0表示邮箱更新密码成功；result值为-1表示邮箱更新密码失败；result值为2表示无法获取当前登录用户的UserID，可能是SDK Token没有登录态；result值为3表示当前登陆账号没有绑定邮箱，无法更新密码。

(11) 发【忘记密码】的手机验证码

- 使用方式：

```
val result  =
CUGEAndroidSDK.authentication().sendForgetPasswordMobileVerifyCode(mobile:
String)
```

- 参数说明：String类型的手机号
- 返回值说明：该函数返回一个Int类型的值，result值为0表示验证码发送成功；result值为-1表示验证码发送失败。

(12) 发【忘记密码】的邮箱验证码

- 使用方式：

```
 val result  =
CUGEAndroidSDK.authentication().sendForgetPasswordEmailVerifyCode(email:
String)
```

- 参数说明：String类型的邮箱账号
- 返回值说明：该函数返回一个Int类型的值，result值为0表示验证码发送成功；result值为-1表示验证码发送失败。

(13) 【手机验证】重置密码

- 使用方式：

```
val mobileRegisterReqBody = MobileRegisterReqBody(mobile: String, password:
String, verifyCode: String)
val result  =
CUGEAndroidSDK.authentication().mobileResetPassword(mobileRegisterReqBody)
```

- 参数说明：先用String类型的手机号、String类型的密码、String类型的验证码构建MobileRegisterReqBody对象，之后将MobileRegisterReqBody对象作为参数传进mobileResetPassword方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示手机号重设密码成功；result值为-1表示手机号重设密码失败；result值为2表示验证码不正确。

(14) 【邮箱验证】重置密码

- 使用方式：

```
val emailRegisterReqBody = EmailRegisterReqBody(email: String, password:
String, verifyCode: String)
val result  =
CUGEAndroidSDK.authentication().emailResetPassword(emailRegisterReqBody)
```

- 参数说明：先用String类型的邮箱账号、String类型的密码、String类型的验证码构建EmailRegisterReqBody对象，之后将EmailRegisterReqBody对象作为参数传进emailResetPassword方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示邮箱账号重设密码成功；result值为-1表示邮箱账号重设密码失败；result值为2表示验证码不正确。

(15) 发【绑定手机】的手机验证码

- 使用方式：

```
val result =
CUGEAndroidSDK.authentication().sBindingMobileVerifyCode(mobile: String)
```

- 参数说明：String类型的手机号
- 返回值说明：该函数返回一个Int类型的值，result值为0表示验证码发送成功；result值为1表示发送验证码过程失败；result值为2表示目标手机号已经被人绑定过了；result值为3表示服务端无法获取正确的登录态；result值为4表示当前登陆的用户已经绑定过手机号，不允许再次绑定；result值为-1表示绑定手机号的验证码发送失败。

(16) 发【绑定邮箱】的邮箱验证码

- 使用方式：

```
val result = CUGEAndroidSDK.authentication().sBindingEmailVerifyCode(email:
String)
```

- 参数说明：String类型的邮箱账号
- 返回值说明：该函数返回一个Int类型的值，result值为0表示验证码发送成功；result值为1表示发送验证码过程失败；result值为2表示目标邮箱已经被人绑定过了；result值为3表示服务端无法获取正确的登录态；result值为4表示当前登陆的用户已经绑定过邮箱，不允许再次绑定；result值为-1表示绑定邮箱的验证码发送失败。

(17) 绑定手机

- 使用方式：

```
val mobileRegisterReqBody = MobileRegisterReqBody(mobile: String, password:
String, verifyCode: String)
val result  =
CUGEAndroidSDK.authentication().bindingMobile(mobileRegisterReqBody)
```

- 参数说明：先用String类型的手机号、String类型的密码、String类型的验证码构建MobileRegisterReqBody对象，之后将MobileRegisterReqBody对象作为参数传进bindingMobile方法
- 返回值说明：该函数返回一个Int类型的值，result值为0表示绑定手机成功；result值为-1表示绑定手机失败；result值为3表示验证码错误或UserID不对应。

(18) 绑定邮箱

- 使用方式：

```
val emailRegisterReqBody = EmailRegisterReqBody(email: String, password:
String, verifyCode: String)
val result  =
CUGEAndroidSDK.authentication().bindingEmail(emailRegisterReqBody)
```

- 参数说明：先用String类型的邮箱账号、String类型的密码、String类型的验证码构建EmailRegisterReqBody对象，之后将EmailRegisterReqBody对象作为参数传进bindingEmail方法。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示绑定邮箱成功；result值为-1表示绑定邮箱失败；result值为3表示验证码错误或UserID不对应。

（19）登录态维持状态下实现自动登录（免登陆）

- 使用方式：

```
if (CUGEAndroidSDK.authentication.isLoginStateExist()) {
       startMainActivity()//免登陆直接进入主界面
 }
```

- 使用说明：调用CUGEAndroidSDK.authentication.isLoginStateExist(),若为true，说明登录态存在，可直接免登陆进入主界面。

2. 业务线

（1）读业务线扩展字段元数据

- 使用方式：

```
val result = CUGEAndroidSDK.businessLine().getBusinessLineMetadata()
```

- 返回值说明：该函数返回一个List数组，数组里的元素为MetadataField对象，MetadataField类的实现如下：

  ```
  data class MetadataField(val name: String, val type: String, val
  isNotNull: Boolean)
  ```

（2）读业务线扩展字段值

- 使用方式：

```
val result = CUGEAndroidSDK.businessLine().getBusinessLineExtFields()
```

- 返回值说明：该函数返回一个List数组，数组里的元素为ExtField对象，ExtField类的实现如下：

  ```
  data class ExtField(val fieldName: String, val value: String)
  ```

（3）修改业务线扩展字段值

- 使用方式：

```
val list = listOf(ExtField(fieldName: String, value: String))
val result = CUGEAndroidSDK.businessLine().modifyBusinessLineExtFields(list)
```

- 参数说明：先构建类型为ExtField的list数组，然后将这个构建好的list数组作为参数传进modifyBusinessLineExtFields中。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示修改业务线扩展字段成功；result值为3表示修改业务线扩展字段失败。

3. 用户信息

（1）查询用户基本信息

- 使用方式：

```
val result = CUGEAndroidSDK.userInfo().queryUserInfo()
```

- 返回值说明：该函数返回一个User类型的对象，通过该对象可以得到userId、nickname等用户的基本信息。

（2）更新用户信息

- 使用方式：

```
val user = User(userId: Int, nickname: String, sex: Boolean, bio: String,
email: String, mobile: String?, birthday: String, avatar: String)
val result = CUGEAndroidSDK.userInfo().updateUserInfo(user)
```

- 参数说明：先构建一个User对象，然后将User对象作为参数传进updateUserInfo中。
- 返回值说明：该函数返回一个Int类型的值，result值为0表示用户信息更新成功；result值为3表示用户信息更新失败。

4. 埋点 （建议先看"埋点说明文档"）

（1）手动上报埋点

- 使用方式：

```
val trackReqBody = TrackUtil.generateTrackBody(trackId: Int)
val tp1 = TrackUtil.generateTrackParameter(endPage: String, parameter:
String)
val tp2 = TrackUtil.generateTrackParameter(startPage: String, widget:
String, parameter: String)
val tp3 = TrackUtil.generateTrackParameter(startPage: String, widget:
String,  endPage: String,  parameter: String)
val result = CUGEAndroidSDK.tracker().track(trackReqBody: TrackReqBody, tp:
String, extData: Any?)
```

- 使用说明：根据埋点事件对应的Id构建TrackReqBody对象，再根据需要构建三种埋点参数中的一种，track函数接收三个参数，第一个参数是TrackReqBody对象，第二个参数是埋点参数，第三个是埋点扩展数据，可为null。
- 返回值说明：该函数没有返回值，在需要手动埋点的地方直接传参数调用即可。

（2）自动化上报埋点RecyclerView

- 使用方式（结合以下示例代码看）

  1）自定义的MyAdapter的构造函数除了传入需要的业务数据（如下面代码中的heroList）外，还需传入AutoTrackConfig对象，然后继承自SDK提供的TrackerAdapter，并将AutoTrackConfig对象作为参数传递给TrackerAdapter的构造函数。

2）内部类ViewHolder需要继承自SDK提供的BannerTrackerViewHolder或者ClickableTrackerViewHolder(根据recyclerView类型选择，BannerTrackerViewHolder是不可点击的banner位曝光，ClickableTrackerViewHolder是可点击的item曝光)

3）构造adapter对象后，不要调用recyclerView的setAdapter方法，而是调用SDK封装后的adapter.adaptWith(recyclerView)，以便内部的埋点跟踪器能够跟踪ViewItem的埋点动作。

```
class MyAdapter(private val heroList: List<String>, config: AutoTrackConfig)
:
    TrackerAdapter<MyAdapter.ViewHolder>(config) {

    inner class ViewHolder(view: View) : ClickableTrackerViewHolder(view,
"pvp2") {
        val heroName: TextView = view.findViewById(R.id.heroName)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.hero_item, parent,
false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val hero = heroList[position]
        holder.heroName.text = hero
        holder.eventParams = "$position"
        holder.extData = hero
    }

    override fun getItemCount() = heroList.size
}

val adapter = MyAdapter(heroList, AutoTrackConfig("MainActivity",
"recyclerView"))
adapter.adaptWith(recyclerView)
```

三、可插拔的QQ登录模块

1. 模块build.gradle中引入依赖：

```
implementation 'com.zzm.qq_auth_library:cuge_qq_auth:1.0.0'
```

2. 具体接入方法

   - 使用方式（结合代码示例看）：

（1）LoginActivity需要继承sdk提供的 QQLoginActivity()。

（2）qq登录点击事件的绑定：调用CUGEAndroidSDK.authentication.qqLogin方法，参数传入QQLoginActivity上下文，注意在协程作用域里调用此suspend方法。

   - 代码示例：

```
class LoginActivity : QQLoginActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        bindComponents()
    }

    private fun bindComponents() {
        login_qq_login.setOnClickListener {
              CUGEAndroidDemoApplication.scope.launch {
                  val result =
CUGEAndroidSDK.authentication.qqLogin(this@LoginActivity)
                  if (result == 0) {
                     "登录成功".showToast()
                  } else {
                      "登录失败".showToast()
                  }
              }
        }
    }
}
```