FeasyBeacon SDK Android 接入文档

Table of contents

[English]

对应 SDK 版本	编写日期	编写人员
V3.3.6	2024 年 11 月 19 日	常纪刚

Chapter 1

SDK 功能概述

1.1 Android 集成

在 Android 应用中集成该 SDK 可快速实现如下功能：

- Beacon 模块的扫描、停止扫描
- Beacon 模块的连接、断开连接
- Beacon 模块信息查询、参数修改
- Beacon 模块空中升级

1.2 了解实例对象

1.2.1 BluetoothDeviceWrapper 类包装了蓝牙设备的信息，用于表示扫描到的蓝牙设备

```
// BluetoothDeviceWrapper 类包装了蓝牙设备的信息，用于表示扫描到的蓝牙设备。
public class BluetoothDeviceWrapper implements Serializable {

  // Beacon 类型常量
  public static final int EDDYSTONE_TYPE = 0x16;
  public static final int ALTBEACON_TYPE = 0xFF;
  public static final int COMPLETE_LOCAL_NAME = 0x09;
```

(continues on next page)

(continued from previous page)

```
    private String address; // 蓝牙设备地址
    private String name;  // 蓝牙设备名称
    private Integer rssi; // 信号强度
    private long timestampNanos = 0; // 时间戳

    // Beacon 对象
    private IBeacon iBeacon = null;
    private EddystoneBeacon eddystoneBeacon = null;
    private AltBeacon altBeacon = null;
    private FeasyBeacon feasyBeacon = null;

    private String completeLocalName = null; // 完整本地名称

    // 监听器对象
    private Monitor monitor = null;

    // 构造函数
    public BluetoothDeviceWrapper(String address) {
        this.address = address;
    }

    // 构造函数
    public BluetoothDeviceWrapper(String address, String name, int␣
↪rssi) {
        this.address = address;
        this.name = name;
        this.rssi = rssi;
    }

    // 构造函数
    public BluetoothDeviceWrapper(String address, String name, int␣
↪rssi, long timestampNanos) {
        this.address = address;
        this.name = name;
        this.rssi = rssi;
```

(continues on next page)

(continued from previous page)

```
    this.timestampNanos = timestampNanos;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public Integer getRssi() {
    return rssi;
}

public void setRssi(Integer rssi) {
    this.rssi = rssi;
}

public long getTimestampNanos() {
    return timestampNanos;
}

public void setTimestampNanos(long timestampNanos) {
    this.timestampNanos = timestampNanos;
}
```

(continues on next page)

(continued from previous page)

```
public IBeacon getIBeacon() {
    return iBeacon;
}

public void setIBeacon(IBeacon iBeacon) {
    if (iBeacon != null) {
        this.iBeacon = iBeacon;
        this.eddystoneBeacon = null;
        this.altBeacon = null;
    }
}

public EddystoneBeacon getEddystoneBeacon() {
    return eddystoneBeacon;
}

public void setEddystoneBeacon(EddystoneBeacon eddystoneBeacon) {
    if (eddystoneBeacon != null) {
        this.eddystoneBeacon = eddystoneBeacon;
        this.iBeacon = null;
        this.altBeacon = null;
    }
}

public AltBeacon getAltBeacon() {
    return altBeacon;
}

public void setAltBeacon(AltBeacon altBeacon) {
    if (altBeacon != null) {
        this.altBeacon = altBeacon;
        this.iBeacon = null;
        this.eddystoneBeacon = null;
    }
}
```

(continues on next page)

(continued from previous page)

```
    public FeasyBeacon getFeasyBeacon() {
        return feasyBeacon;
    }

    public void setFeasyBeacon(FeasyBeacon feasyBeacon) {
        this.feasyBeacon = feasyBeacon;
    }

    public String getCompleteLocalName() {
        return completeLocalName;
    }

    public void setCompleteLocalName(String completeLocalName) {
        this.completeLocalName = completeLocalName;
    }

    public Monitor getMonitor() {
        return monitor;
    }

    public void setMonitor(Monitor monitor) {
        this.monitor = monitor;
    }
}
```

1.2.2 EddystoneBeacon 类表示 Eddystone 协议的 Beacon

```
// EddystoneBeacon 类表示 Eddystone 协议的 Beacon
public class EddystoneBeacon extends FeasyBeacon {

    private final String TAG = "FscEddystone";
    private String frameTypeString; // Eddystone 类型（URL、UID、TLM）
    private String frameTypeHex; // Eddystone 类型的十六进制表示
    private int eddystoneRssiOrVersion; // Eddystone 的发射功率或版本号
```

(continues on next page)

(continued from previous page)

```
private String dataValue; // Eddystone 的广播数据

private String url;
private String nameSpace;
private String instance;
private String reserved;

private String batteryVoltage;
private String temperature;
private String advertisementsCount;
private String timeSincePowerUp;

public int getEddystoneRssiOrVersion() {
    return eddystoneRssiOrVersion;
}

public void setEddystoneRssiOrVersion(int eddystoneRssiOrVersion) {
    this.eddystoneRssiOrVersion = eddystoneRssiOrVersion;
}

public String getFrameTypeString() {
    return frameTypeString;
}

public void setFrameTypeString(String frameTypeString) {
    this.frameTypeString = frameTypeString;
}

public String getFrameTypeHex() {
    return frameTypeHex;
}

public void setFrameTypeHex(String frameTypeHex) {
    if (frameTypeHex.length() == 1) {
        frameTypeHex = "0" + frameTypeHex;
```

(continues on next page)

(continued from previous page)

```
        }
        this.frameTypeHex = frameTypeHex;
    }

    public String getDataValue() {
        return dataValue;
    }

    public void setDataValue(String dataValue) {
        this.dataValue = dataValue;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getNameSpace() {
        return nameSpace;
    }

    public void setNameSpace(String nameSpace) {
        this.nameSpace = nameSpace;
    }

    public String getInstance() {
        return instance;
    }

    public void setInstance(String instance) {
        this.instance = instance;
    }
```

(continues on next page)

(continued from previous page)

```
public String getReserved() {
    return reserved;
}

public void setReserved(String reserved) {
    this.reserved = reserved;
}

public String getBatteryVoltage() {
    return batteryVoltage;
}

public void setBatteryVoltage(String batteryVoltage) {
    this.batteryVoltage = batteryVoltage;
}

public String getTemperature() {
    return temperature;
}

public void setTemperature(String temperature) {
    this.temperature = temperature;
}

public String getAdvertisementsCount() {
    return advertisementsCount;
}

public void setAdvertisementsCount(String advertisementsCount) {
    this.advertisementsCount = advertisementsCount;
}

public String getTimeSincePowerUp() {
    return timeSincePowerUp;
```

(continues on next page)

(continued from previous page)

```
    }

    public void setTimeSincePowerUp(String timeSincePowerUp) {
        this.timeSincePowerUp = timeSincePowerUp;
    }
}
```

1.2.3 AltBeacon 表示 AltBeacon 类型的蓝牙信标

```
// AltBeacon 类表示 AltBeacon 类型的蓝牙信标
public class AltBeacon extends FeasyBeacon {

    private String reservedId; // 保留 ID
    private String manufacturerId; // 制造商 ID
    private String beaconId; // 信标 ID
    private int altBeaconRssi; // AltBeacon 的 RSSI 值
    private String feature; // 特征

    public int getAltBeaconRssi() {
        return altBeaconRssi;
    }

    public void setAltBeaconRssi(int altBeaconRssi) {
        this.altBeaconRssi = altBeaconRssi;
    }

    public String getBeaconId() {
        return beaconId;
    }

    public void setBeaconId(String beaconId) {
        this.beaconId = beaconId;
    }

    public String getManufacturerId() {
```

(continues on next page)

(continued from previous page)

```
        return manufacturerId;
    }

    public void setManufacturerId(String manufacturerId) {
        this.manufacturerId = manufacturerId;
    }

    public String getReservedId() {
        return reservedId;
    }

    public void setReservedId(String reservedId) {
        this.reservedId = reservedId;
    }

    public String getFeature() {
        return feature;
    }

    public void setFeature(String feature) {
        this.feature = feature;
    }
}
```

1.2.4 IBeacon 类表示 iBeacon 协议的信标设备

```
// IBeacon 类表示 iBeacon 协议的信标设备
public class IBeacon extends FeasyBeacon {

    private String company; // 公司标识
    private String type; // 信标类型
    private String dataLength; // 数据长度
    private int major; // Major 值，用于区分信标
    private int minor; // Minor 值，用于区分信标
    private String uuid; // UUID，用于唯一标识信标
```

(continues on next page)

(continued from previous page)

```
private int rssi; // 接收信号强度指示（RSSI）

public String getCompany() {
    return company;
}

public void setCompany(String company) {
    this.company = company;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public String getDataLength() {
    return dataLength;
}

public void setDataLength(String dataLength) {
    this.dataLength = dataLength;
}

public int getMajor() {
    return major;
}

public void setMajor(int major) {
    this.major = major;
}

public int getMinor() {
```

(continues on next page)

(continued from previous page)

```
        return minor;
    }

    public void setMinor(int minor) {
        this.minor = minor;
    }

    public String getUuid() {
        return uuid;
    }

    public void setUuid(String uuid) {
        this.uuid = uuid;
    }

    public int getRssi() {
        return rssi;
    }

    public void setRssi(int rssi) {
        this.rssi = rssi;
    }
}
```

1.2.5 Monitor 类表示一种监控器设备，包括其温度、湿度、电池电量和名称属性

```
// Monitor 类表示一种监控器设备，包括其温度、湿度、电池电量和名称属性
public class Monitor extends FeasyBeacon {

    private String temperature; // 温度值
    private String humidity; // 湿度值
    private int battery; // 电池电量
    private String name; // 监控器名称
```

(continues on next page)

(continued from previous page)

```
    public String getTemperature() {
        return temperature;
    }

    public void setTemperature(String temperature) {
        this.temperature = temperature;
    }

    public String getHumidity() {
        return humidity;
    }

    public void setHumidity(String humidity) {
        this.humidity = humidity;
    }

    @Override
    public int getBattery() {
        return battery;
    }

    @Override
    public void setBattery(int battery) {
        this.battery = battery;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

1.2.6 FeasyBeacon 类表示一个信标设备，包括其相关的属性和配置信息

```
// FeasyBeacon 类表示一个信标设备，包括其相关的属性和配置信息
public class FeasyBeacon implements Serializable {

  // Beacon 类型常量
  public static final String BEACON_TYPE_EDDYSTONE_URL = "URL";
  public static final String BEACON_TYPE_EDDYSTONE_UID = "UID";
  public static final String BEACON_TYPE_EDDYSTONE_TLM = "TLM";
  public static final String BEACON_TYPE_I_BEACON = "iBeacon";
  public static final String BEACON_TYPE_ALTBEACON = "AltBeacon";
  public static final String BEACON_TYPE_NULL = "";
  public static final int BEACON_COUNT = 10;
  public static final String ENCRYPT_BEACON = "Beacon";
  public static final String ENCRYPT_UNIVERSAL = "Universal";

  private boolean keyConfig; // 配置键
  private boolean gSensor; // 配置传感器
  private boolean buzzer; // 配置蜂鸣器
  private boolean led; // 配置 LED
  private boolean nfc; // 配置 NFC
  private boolean longRange; // 配置长距离
  private boolean connectable; // 是否可连接
  private String module; // 模块类型
  private String version; // 版本信息
  private String deviceName; // 设备名称
  private String deviceAddress; // 设备地址
  private int battery = -1; // 电池电量
  private String encryptionWay; // 加密方式

  public int getBattery() {
      return battery;
  }

  public void setBattery(int battery) {
```

(continues on next page)

(continued from previous page)

```
    this.battery = battery;
}

public boolean getConnectable() {
    return connectable;
}

public void setConnectable(boolean connectable) {
    this.connectable = connectable;
}

public String getVersion() {
    return version;
}

public void setVersion(String version) {
    this.version = version;
}

public String getDeviceName() {
    return deviceName;
}

public void setDeviceName(String deviceName) {
    this.deviceName = deviceName;
}

public String getDeviceAddress() {
    return deviceAddress;
}

public void setDeviceAddress(String deviceAddress) {
    this.deviceAddress = deviceAddress;
}
```

(continues on next page)

(continued from previous page)

```
public String getModule() {
    return module;
}

public void setModule(String module) {
    this.module = module;
}

public String getEncryptionWay() {
    return encryptionWay;
}

public void setEncryptionWay(String encryptionWay) {
    this.encryptionWay = encryptionWay;
}

public boolean getKeyConfig() {
    return keyConfig;
}

public void setKeyConfig(boolean keyConfig) {
    this.keyConfig = keyConfig;
}

public boolean getGSensor() {
    return gSensor;
}

public void setGSensor(boolean gSensor) {
    this.gSensor = gSensor;
}

public boolean getBuzzer() {
    return buzzer;
}
```

(continues on next page)

(continued from previous page)

```
    public void setBuzzer(boolean buzzer) {
        this.buzzer = buzzer;
    }

    public boolean getLed() {
        return led;
    }

    public void setLed(boolean led) {
        this.led = led;
    }

    public boolean getNfc() {
        return nfc;
    }

    public void setNfc(boolean nfc) {
        this.nfc = nfc;
    }

    public boolean isLongRange() {
        return longRange;
    }

    public void setLongRange(boolean longRange) {
        this.longRange = longRange;
    }
}
```

1.2.7 BeaconBean 类用于表示蓝牙信标 (包含了 iBeacon、eddystone_url、eddystone_uid、altBeacon) 的属性信息

```
// BeaconBean 类用于表示蓝牙信标的属性信息
public class BeaconBean implements Serializable {
```

(continues on next page)

(continued from previous page)

```
    private String index; // 索引
    private String beaconType; // 信标类型 (iBeacon, Eddystone_url,␣
↪Eddystone_uid)

    //iBeacon
    private String uuid; // iBeacon 的 UUID
    private String major; // iBeacon 的 Major 值
    private String minor;// iBeacon 的 Minor 值

    //eddystone_url
    private String url; // Eddystone_url 的 URL

    //eddystone_uid
    private String nameSpace; // Eddystone_uid 的命名空间
    private String instance; // Eddystone_uid 的实例
    private String reserved; // EddyStoner 的保留字段

    //altBeacon
    private String id1; // AltBeacon 的 ID1
    private String id2; // AltBeacon 的 ID2
    private String id3; // AltBeacon 的 ID3
    private String manufacturerId; // 制造商 ID
    private String manufacturerReserved; // 制造商保留字段

    private String power; // 信标 TX 功率
    private boolean connectable; // 是否可连接
    private boolean enable; // 广播是否使能
    private String version;// 固件版本
    private int interval = 1000; // 信标间隔，默认值为 1000ms
    private String parentName; // 父节点名称
    private String txpower = "2"; // 信标 TX 功率，默认值为 2
    private String phy = "0"; // 物理层，默认值为 0

    public BeaconBean() {
    }
```

(continues on next page)

(continued from previous page)

```
public BeaconBean(String index, String beaconType) {
    this.index = index;
    this.beaconType = beaconType;
}

public String getIndex() {
    return index;
}

public void setIndex(String index) {
    this.index = index;
}

public String getBeaconType() {
    return beaconType;
}

public void setBeaconType(String beaconType) {
    this.beaconType = beaconType;
}

public String getUuid() {
    return uuid;
}

public void setUuid(String uuid) {
    this.uuid = uuid;
}

public String getMajor() {
    return major;
}

public void setMajor(String major) {
```

(continues on next page)

(continued from previous page)

```
        this.major = major;
    }

    public String getMinor() {
        return minor;
    }

    public void setMinor(String minor) {
        this.minor = minor;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getNameSpace() {
        return nameSpace;
    }

    public void setNameSpace(String nameSpace) {
        this.nameSpace = nameSpace;
    }

    public String getInstance() {
        return instance;
    }

    public void setInstance(String instance) {
        this.instance = instance;
    }
```

(continues on next page)

(continued from previous page)

```
public String getReserved() {
    return reserved;
}

public void setReserved(String reserved) {
    this.reserved = reserved;
}

public String getId1() {
    return id1;
}

public void setId1(String id1) {
    this.id1 = id1;
}

public String getId2() {
    return id2;
}

public void setId2(String id2) {
    this.id2 = id2;
}

public String getId3() {
    return id3;
}

public void setId3(String id3) {
    this.id3 = id3;
}

public String getManufacturerId() {
    return manufacturerId;
}
```

(continues on next page)

(continued from previous page)

```
public void setManufacturerId(String manufacturerId) {
    this.manufacturerId = manufacturerId;
}

public String getManufacturerReserved() {
    return manufacturerReserved;
}

public void setManufacturerReserved(String manufacturerReserved) {
    this.manufacturerReserved = manufacturerReserved;
}

public String getPower() {
    return power;
}

public void setPower(String power) {
    this.power = power;
}

public boolean isConnectable() {
    return connectable;
}

public void setConnectable(boolean connectable) {
    this.connectable = connectable;
}

public boolean isEnable() {
    return enable;
}

public void setEnable(boolean enable) {
    this.enable = enable;
```

(continues on next page)

(continued from previous page)

```
}

public String getVersion() {
    return version;
}

public void setVersion(String version) {
    this.version = version;
}

public int getInterval() {
    return interval;
}

public void setInterval(int interval) {
    this.interval = interval;
}

public String getParentName() {
    return parentName;
}

public void setParentName(String parentName) {
    this.parentName = parentName;
}

public String getTxpower() {
    return txpower;
}

public void setTxpower(String txpower) {
    this.txpower = txpower;
}

public String getPhy() {
```

(continues on next page)

(continued from previous page)

```
        return phy;
    }

    public void setPhy(String phy) {
        this.phy = phy;
    }
}
```

Chapter 2

快速集成（Android）

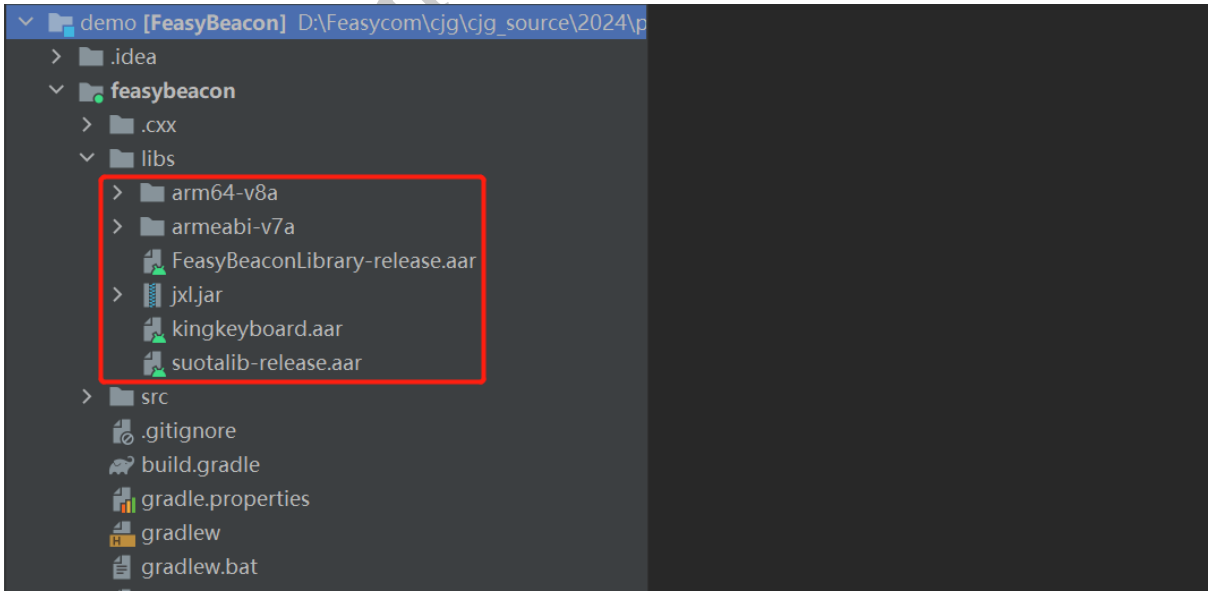
2.1 集成 SDK

- 如果您还未安装 Android Studio，请访问 [安卓官网]

2.1.1 创建 Android 工程

- 在 Android Studio 中新建项目。

2.1.2 将 FeasyBeaconLibrary-release.aar、suotalib-release.aar 和 so 库放到项目 libs 目录



2.1.3 配置 build.gradle 文件

```
dependencies {
  implementation fileTree(include: ['*.jar','*.aar'], dir: 'libs')
}
```

2.1.4 配置 AndroidManifest.xml 文件

- 在您的应用程序中，需要在 AndroidManifest.xml 文件中给予权限，请在您的应用程序清单文件中添加如下代码。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  package="com.feasycom.feasybeacon">

  <!-- 网络和定位权限 -->
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_
↪LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_
↪LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_
↪STATE" />
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE
↪" />
  <uses-permission android:name="android.permission.WAKE_LOCK" />

  <!-- 蓝牙权限 -->
  <uses-permission android:name="android.permission.BLUETOOTH_CONNECT
↪" />
  <uses-permission android:name="android.permission.BLUETOOTH_SCAN" /
↪>
  <!-- 添加蓝牙权限声明 -->
  <uses-permission android:name="android.permission.BLUETOOTH" />
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"␣
↪/>
```

(continues on next page)

(continued from previous page)

```
    <!-- 外部存储权限 -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_
↪STORAGE" />

    <!-- 读写特权权限 -->
    <uses-permission
        android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"
        tools:ignore="ProtectedPermissions" />
    <uses-permission
        android:name="android.permission.WRITE_SECURE_SETTINGS"
        tools:ignore="ProtectedPermissions" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_
↪STORAGE" />

    <uses-permission android:name="android.permission.READ_MEDIA_IMAGES
↪" />

    <uses-permission android:name="android.permission.BLUETOOTH_
↪ADVERTISE" />

    <application
        android:name=".App"
        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/Theme.FeasyBeacon1">

        <!-- 服务 -->
        <service android:name="com.feasycom.service.AtCommandService"␣
↪android:enabled="true" android:exported="false" />
```

(continues on next page)

(continued from previous page)

```
        <service android:name="com.feasycom.service.OTABLEService"␣
↪android:enabled="true" android:exported="false" />

    </application>

</manifest>
```

2.2 接口使用方法示例

2.2.1 负责管理蓝牙操作和相关数据

```
object BluetoothRepository {

    // 选中的蓝牙设备映射
    val selectedDeviceMap = LinkedHashMap<String, Any>()
    // 模块编号
    var mModule = 0
    // 版本范围
    var mVersionRange: Range<Int>? = null
    // 已发现的蓝牙设备列表
    var mDevices = mutableListOf<BluetoothDeviceWrapper>()
    // 用于回调的列表
    private val mCallbacks = mutableListOf<FscBleCallback>()
    // 蓝牙连接 PIN 码
    var mConnectPin = "000000"
    // 指令 Bean
    val commandBean: CommandBean = CommandBean()
    // FscBeacon 中心 API 实例
    private val mFscBeaconCentralApi by lazy {
        FscBeaconApiImp.getInstance(App.sContext).apply {
            initialize()
            setCallbacks(BeaconCallbacks())
        }
    }
```

(continues on next page)

(continued from previous page)

```
    /**
     * 检查蓝牙是否启用
     */
    fun isBluetoothEnabled() = mFscBeaconCentralApi.isBluetoothEnabled

    /**
     * 开始扫描蓝牙设备
     */
    fun startScan() = mFscBeaconCentralApi.startScan()

    /**
     * 开始指定时间的扫描
     * @param scanTime 扫描时间
     */
    fun startScanTime(scanTime: Int) = mFscBeaconCentralApi.
↪startScan(scanTime)

    /**
     * 停止扫描
     */
    fun stopScan() = mFscBeaconCentralApi.stopScan()

    /**
     * 判断设备是否已连接
     */
    fun isConnect() = mFscBeaconCentralApi.isConnected

    /**
     * 连接指定设备
     * @param device 蓝牙设备包装器
     * @param pin PIN 码
     */
    fun connect(device: BluetoothDeviceWrapper, pin: String) =␣
↪mFscBeaconCentralApi.connect(device, pin)
```

(continues on next page)

(continued from previous page)

```
    /**
     * 断开连接
     */
    fun disconnect() = mFscBeaconCentralApi.disconnect()

    /**
     * 判断 Beacon 信息是否完整
     */
    fun isBeaconInfoFull() = mFscBeaconCentralApi.isBeaconInfoFull

    /**
     * 添加 Beacon 信息
     * @param beaconBean BeaconBean 实例
     */
    fun addBeaconInfo(beaconBean: BeaconBean) = mFscBeaconCentralApi.
↪addBeaconInfo(beaconBean)

    /**
     * 删除 Beacon 信息
     * @param beaconBean BeaconBean 实例
     */
    fun deleteBeaconInfo(beaconBean: BeaconBean) {
        mFscBeaconCentralApi.deleteBeaconInfo(beaconBean.index)
        mCallbacks.forEach { it.onDeleteBeaconInfo(beaconBean) }
    }

    /**
     * 设置设备名称
     * @param deviceName 设备名称
     */
    fun setDeviceName(deviceName: String) = mFscBeaconCentralApi.
↪setDeviceName(deviceName)

    /**
     * 设置设备 PIN 码
```

(continues on next page)

(continued from previous page)

```
    * @param pin PIN 码
    */
    fun setDevicePin(pin: String) = mFscBeaconCentralApi.
↪setDevicePin(pin)

    /**
    * 设置广播间隔
    * @param intervalTime 间隔时间
    */
    fun setBroadcastInterval(intervalTime: String) =␣
↪mFscBeaconCentralApi.setBroadcastInterval(intervalTime)

    /**
    * 恢复设备
    */
    fun restore() = mFscBeaconCentralApi.restore()

    /**
    * 查询或设置固件密钥
    */
    fun setFirmwareKey(command : String) = mFscBeaconCentralApi.
↪setFirmwareKey(command)

    /**
    * 设置应用密钥
    */
    fun setAppKey(key: String) = mFscBeaconCentralApi.setAppKey(key)

    /**
    * 获取 Beacon 广播参数
    */
    fun beaconBroadcastParameters() = mFscBeaconCentralApi.
↪beaconBroadcastParameters();

    /**
```

(continues on next page)

(continued from previous page)

```
    * 设置 TLM 状态
    * @param isTlm 是否启用 TLM
    */
    fun setTlm(isTlm: Boolean) = mFscBeaconCentralApi.setTlm(isTlm)

    /**
    * 设置广播扩展
    * @param isBroadcastExtensions 是否启用广播扩展
    */
    fun setBroadcastExtensions(isBroadcastExtensions: Boolean) =␣
↪mFscBeaconCentralApi.setBroadcastExtensions(isBroadcastExtensions)

    /**
    * 检查 DFU 文件
    * @param dfuFile DFU 文件字节数组
    */
    fun checkDfuFile(dfuFile: ByteArray) = mFscBeaconCentralApi.
↪checkDfuFile(dfuFile)!!

    /**
    * 开始 OTA 更新
    * @param dfuFile DFU 文件字节数组
    * @param reset 是否重置
    */
    fun startOtaUpdate(dfuFile: ByteArray, reset: Boolean) =␣
↪mFscBeaconCentralApi.startOtaUpdate(dfuFile, reset)

    /**
    * 设置 Beacon 信息
    */
    fun setBeaconInfo() = mFscBeaconCentralApi.saveBeaconInfo()

    /**
    * 批量设置 Beacon 信息
    */
```

(continues on next page)

(continued from previous page)

```
    fun setBatchBeaconInfo() = mFscBeaconCentralApi.
↪saveBatch(commandBean)

    /**
     * 注册视图回调
     * @param callback 回调接口
     */
    fun registerViewCallback(callback: FscBleCallback) {
        if (mCallbacks.contains(callback)) return
        try {
            mCallbacks.add(callback)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    /**
     * 取消注册视图回调
     * @param callback 回调接口
     */
    fun unRegisterViewCallback(callback: FscBleCallback) {
        try {
            mCallbacks.remove(callback)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    /**
     * 获取设备信息
     * @param feasyBeacon FeasyBeacon 实例
     * @param isTxPower 是否为发射功率
     */
    fun getDeviceInfo(feasyBeacon: FeasyBeacon, isTxPower: Boolean) {
        mFscBeaconCentralApi.getDeviceInfo(feasyBeacon, isTxPower)
```

(continues on next page)

(continued from previous page)

```
    }

    /**
     * 内部回调类，用于处理蓝牙操作的回调事件
     */
    class BeaconCallbacks : FscBeaconCallbacks {

        // 开始扫描
        override fun startScan() {}

        // 停止扫描
        override fun stopScan() {}

        // 连接进度更新
        override fun connectProgressUpdate(device: BluetoothDevice?,␣
↪status: Int) {
            mCallbacks.forEach { it.onConnectProgressUpdate(status) }
        }

        // 发现蓝牙外设
        override fun blePeripheralFound(device: BluetoothDeviceWrapper?
↪, rssi: Int, record: ByteArray?) {
            // 如果设备为空，则返回
            if (device == null)  return

            try {
                // 检查设备是否为 iBeacon、EddystoneBeacon 或 AltBeacon
                if (device.iBeacon != null || device.eddystoneBeacon !
↪= null || device.altBeacon != null) {
                    notifyCallbacks { it.onBeacon(device) }
                }

                // 如果设备包含监控数据
                device.monitor?.let {
                    notifyCallbacks { it.onSensor(device) }
```

(continues on next page)

(continued from previous page)

```
                }

                // 如果设备包含 FeasyBeacon 模块数据
                device.feasyBeacon?.module?.let {
                    notifyCallbacks { it.onSetting(device) }
                }
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }

        // 通知所有回调函数
        private inline fun notifyCallbacks(action: (FscBleCallback) ->␣
↪Unit) {
            mCallbacks.forEach { callback ->
                try {
                    action(callback)
                } catch (e: Exception) {
                    // 捕获异常并打印
                    e.printStackTrace()
                }
            }
        }

        // 蓝牙外设已连接
        override fun blePeripheralConnected(gatt: BluetoothGatt?,␣
↪device: BluetoothDevice?) {
            mCallbacks.forEach { it.onConnectedSuccess() }
        }

        // 蓝牙外设断开连接
        override fun blePeripheralDisconnected(gatt: BluetoothGatt?,␣
↪device: BluetoothDevice?) {
            mCallbacks.forEach { it.onDisconnect() }
        }
```

(continues on next page)

(continued from previous page)

```
    // AT 命令回调
    override fun atCommandCallBack(command: String?, param: String?
↪, status: String?) {
        mCallbacks.forEach { it.onAtCommandCallBack(command, param,
↪ status) }
    }

    // 发现服务
    override fun servicesFound(
        gatt: BluetoothGatt?,
        device: BluetoothDevice?,
        services: ArrayList<BluetoothGattService>?
    ) {}

    // 服务中发现特性
    override fun characteristicForService(
        gatt: BluetoothGatt?,
        device: BluetoothDevice?,
        service: BluetoothGattService?,
        characteristic: BluetoothGattCharacteristic?
    ) {}

    // 接收到数据包
    override fun packetReceived(
        gatt: BluetoothGatt?,
        device: BluetoothDevice?,
        service: BluetoothGattService?,
        ch: BluetoothGattCharacteristic?,
        strValue: String,
        hexString: String,
        rawValue: ByteArray,
        timestamp: String?
    ) {
        mCallbacks.forEach { it.onPacketReceived(strValue,␣
```

(continues on next page)

(continued from previous page)

```
↪hexString, rawValue) }
    }

    // 接收到读取响应
    override fun readResponse(
        gatt: BluetoothGatt?,
        device: BluetoothDevice?,
        service: BluetoothGattService?,
        ch: BluetoothGattCharacteristic?,
        strValue: String?,
        hexString: String?,
        rawValue: ByteArray?,
        timestamp: String?
    ) {}

    // 发送数据包进度
    override fun sendPacketProgress(
        gatt: BluetoothGatt?,
        device: BluetoothDevice?,
        ch: BluetoothGattCharacteristic?,
        percentage: Int,
        sendByte: ByteArray?
    ) {}

    // OTA 进度更新
    override fun otaProgressUpdate(percentage: Int, status: Int) {
        mCallbacks.forEach { it.onOtaProgressUpdate(percentage) }
    }

    // 接收到设备信息
    override fun onDeviceInfoReceived(parameterName: String?,␣
↪parameter: Any?) {
        mCallbacks.forEach { it.onDeviceInfo(parameterName,␣
↪parameter) }
    }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

2.2.2 SUOTA SDK 升级使用示例

```
class SuotaActivity : BaseActivity<ActivitySuotaBinding>(),
DeviceInfoFragment.OnDeviceInfoFragmentInteractionListener,
AvailableFirmwareFragment.
↪OnAvailableFirmwareFragmentInteractionListener,
UpdateFirmwareFragment.OnUpdateFirmwareFragmentInteractionListener,
BaseSuotaFragment.OnBaseSuotaFragmentInteractionListener{

    private lateinit var suotaFile: SuotaFile
    private lateinit var suotaManager: SuotaManager

    // 建立 GATT 连接
    override fun connect() {
        // 初始化 SuotaManager
        suotaManager = SuotaManager(lifecycle, this, bluetoothDevice,
↪ SuotaCallback(this)).apply {
            setUiContext(this@SuotaActivity)
            connect()
        }
    }

    // 设置升级文件
    override fun onFirmwareSelected(suotaFile: SuotaFile?) {
        suotaManager.suotaFile = suotaFile
        suotaManager.initializeSuota(240, 3, 0, 1, 4, 2)
    }

    // 开始固件升级
    override fun startSuotaProtocol() {
        suotaManager.startUpdate()
    }
```

(continues on next page)

(continued from previous page)

```
}

// 一个用于处理 SUOTA（无线固件升级）事件的回调类。
class SuotaCallback(@NonNull private val suotaActivityRef:␣
↪SuotaActivity): SuotaManagerCallback() {

    // 当连接状态发生变化时调用。
    override fun onConnectionStateChange(newStatus: SuotaProfile.
↪SuotaManagerStatus) {

    }

    // 当发现服务时调用。
    override fun onServiceDiscovered() {

    }

    // 当读取特征值时调用。
    override fun onCharacteristicRead(characteristicGroup:␣
↪SuotaProfile.CharacteristicGroup, characteristic:␣
↪BluetoothGattCharacteristic) {

    }

    // 当读取设备信息完成时调用。
    override fun onDeviceInfoReadCompleted(status: SuotaProfile.
↪DeviceInfoReadStatus) {

    }

    // 当设备准备好进行 SUOTA 时调用。
    override fun onDeviceReady() {

    }
```

(continues on next page)

(continued from previous page)

```
    // 记录 SUOTA 状态和附加信息。
    override fun onSuotaLog(state: SuotaProfile.SuotaProtocolState,␣
↪type: ISuotaManagerCallback.SuotaLogType, log: String) {

    }

    // 在分块数据传输过程中调用。
    override fun onChunkSend(chunkCount: Int, totalChunks: Int,␣
↪chunk: Int, block: Int, blockChunks: Int, totalBlocks: Int) {

    }

    // 在上传进度发生变化时更新进度条。
    override fun onUploadProgress(percent: Float) {

    }

    // 当 SUOTA 过程成功完成时调用。
    override fun onSuccess(totalElapsedSeconds: Double,␣
↪imageUploadElapsedSeconds: Double) {

    }

    // 当 SUOTA 过程失败时调用。
    override fun onFailure(errorCode: Int) {

    }

    // 当发送重启命令时调用。
    override fun onRebootSent() {

    }

    // 在上传过程中更新速度统计信息。
```

(continues on next page)

(continued from previous page)

```
    override fun updateSpeedStatistics(current: Double, max: Double,␣
↪min: Double, avg: Double) {

    }

    // 在上传过程中更新当前速度。
    override fun updateCurrentSpeed(currentSpeed: Double) {

    }

    // 处理显示挂起的重启对话框。
    override fun pendingRebootDialog(rebootDialog:␣
↪SupportCustomDialogFragment) {

    }

}
```

Chapter 3

附录

3.1 下载 PDF 版本

下载 PDF 版本