布丁豆场景接入文档

版本	日期	说明	作者
1.0	20171031	init	Lee
1.1	20171103	修改错误	Lee

概念

Scene（场景），在产品层面上是体现一个机器人交互能力的直接途径，是体现用户体验质量的直接载体，是用户的直接交互对象，比如翻译场景；同时在技术层面上是AI消息传递机制中的module（基本模块），是AICommand发送的目标(target)，场景作为与用户直接交互的对象，拥有自己的生命周期，开发者用户需要严格按照生命周期来开发自己的程序。

场景名

场景名即 module name（模块名），当系统收到的指令总会有一个 target（发送目标），它往往指的就是 module name（模块名）。

顶级场景

在模块通信机制中，如果指令目标（target）是一个场景，那么系统将总是会先将场景启动并置为顶级场景，场景与场景之间是互斥的，顶级场景只有一个，换句话说一个场景只有是顶级场景的时候才能接收AICommand（指令）。

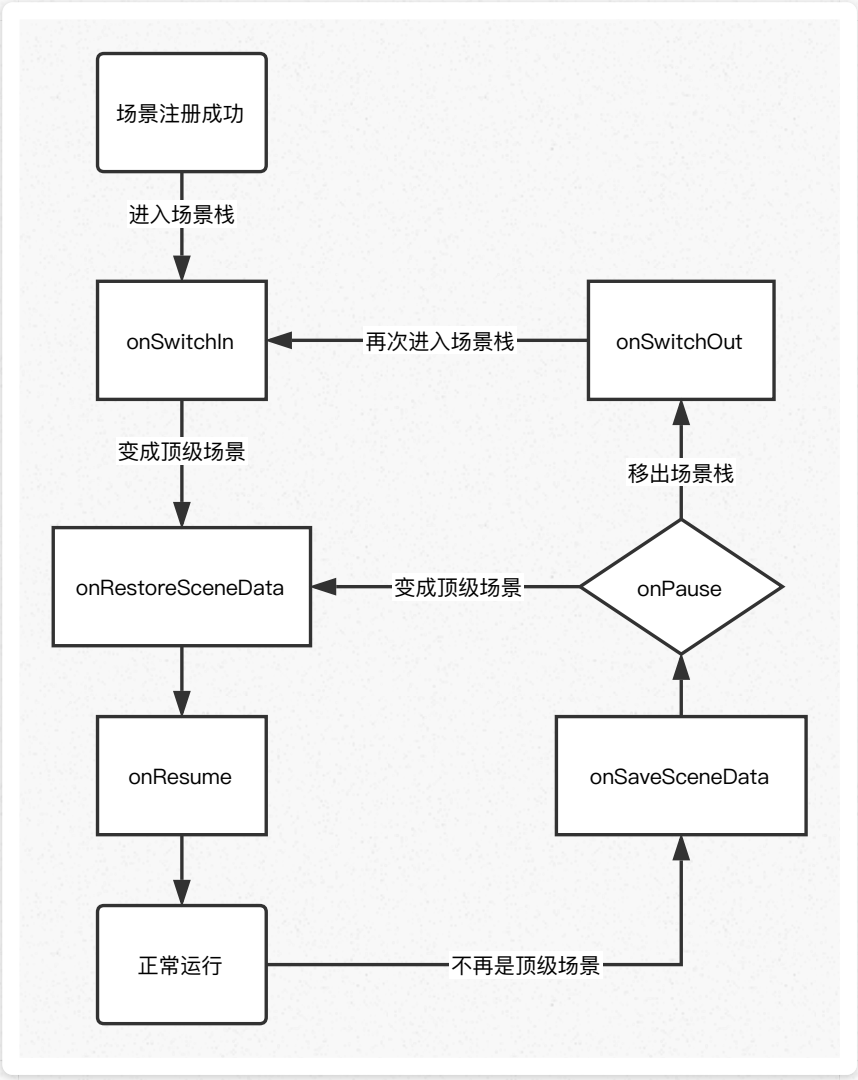
生命周期函数

一个 Scene（场景）的生命周期函数有如下几个：
* onSwitchIn： 场景被切入。场景的切入指的是场景被放入场景栈。
* onRestoreSceneData：恢复场景数据。这个回调只有在onSaveSceneData中场景确实存入了数据之后才会被调用。
* onResume：场景被恢复。
* onPause：场景被暂停。
* onSaveSceneData：保存场景数据。
* onSwitchOut：场景被切出。

场景的声明周期和场景栈密切相关。场景栈是在系统内维护的一个运行中的场景的栈。默认情况下，场景栈是空的。当一个场景被调起和用户交互的时候就会被放入场景栈。根据新启动的场景以及上一个场景的类型，上一个场景可能被留在场景栈当中，也有可能被清除出场景栈。当场景被放入场景栈的时候onSwitchIn会被调用，当场景被清除出场景栈的时候onSwitchOut会被调用。场景的切入切出以及运行状态与场景APP的生命周期没有关系。

当场景没有被清除出场景栈，此时有另一个场景变成顶级场景，那么原先的场景就会收到onSaveSceneData和onPause。场景可以选择将数据保存下来。之后，如果该场景又变成顶级场景，就会收到onRestoreSceneData和onResume。
onRestoreSceneData并不总是被调用的，如果在onSaveSceneData的时候没有存入任何数据，就不会收到回调。

相关的生命周期函数被调用的时机见下图:



相关接口定义具体参看 com.roobo.core.scene.SceneEventListener

注意：场景的 onSwitchIn、onCommand、onSwitchOut 函数均是运行在 UI 线程。

消息接收回调

当场景成为顶级场景后，就可以开始接收指令（AICommand）了，对应 onCommand 方法，该方法有三个参数，具体含义在用法中将详细说明。相关接口定义具体参看 com.roobo.core.scene.SceneEventListener。

场景切换类型

场景可以按照切换的行为分为普通场景，层叠场景和持续场景。这种类型与场景在场景栈中的行为有关。
* 普通场景：对于一个普通场景，当它被切入场景栈的时候会将原先的场景踢出场景栈。普通场景会尽量要求场景栈中只有它自己。系统被唤醒或者打断的时候，如果场景栈中包含普通场景，那么这个场景会被切出。

- 层叠场景：层叠场景启动的时候不会要求上一个场景退出场景栈。层叠场景的切入会导致之前的场景被暂停，对应的会收到onPause回调。当层叠场景切出时，场景栈中的上一个场景会收到onResume回调。系统被唤醒或者打断的时候，层叠场景会被切出。
- 持续场景。持续场景除非自己调用switchOut，永远都不会切出。当有另外一个场景（无论是普通场景还是层叠场景）切入时，持续场景会收到onPause回调，当上一个场景被切出，持续场景会收到onResume回调。系统被唤醒或者打断只会让持续场景暂停，不会把持续场景切出场景栈。
- 打断的行为。打断指的是用户在系统处于唤醒状态的时候说出唤醒词，导致系统当前进行的行为被终止。对于场景栈来说，打断会导致场景栈从栈顶开始清理场景，直到遇到持续场景或者所有场景都被清理。

场景切换类型的flag请参考 com.roobo.core.scene.SceneEventListener

接入sdk

SceneDemo 是接入场景的示例，包含接入场景以及布丁豆提供的常用功能。

声明 - AndroidManifest.xml

一个场景在开发前需要在 AndroidManifest.xml 中声明两个 meta-data 段，如下例子中，场景 Translator 声明：

```
<meta-data
    android:name="ROOBO_MODULE_NAME"
    android:value="Translator" />
<meta-data
    android:name="ROOBO_MODULE_TYPE"
    android:value="scene"/>
```

其中 ROOBO_MODULE_NAME 代表该模块的名字，必须声明，同一系统上的模块名不能重复，ROOBO_MODULE_TYPE 代表该模块的类型，值有三种，scene、service和其他，默认不写该项指的是其他。作为一个场景，值为 scene。 一个场景开发时可以选择依赖 **SceneSDK.jar**，也可以选择依赖我们提供的 aar，如果是前者，则还需要在 AndroidManifest.xml 中声明一个 service：

```
<service
    android:name="com.roobo.core.communication.DefaultService"
    android:exported="true"/>
```

保活机制

如果希望进程在后台不被杀死，可在AndroidManifest.xml里添加如下配置

```
<meta-data
            android:name="ROOBO_KEEP_ALIVE"
            android:value="true"/>
```

初始化 - Application.onCreate()

场景的初始化不同于模块初始化，它依赖 com.roobo.core.scene.SceneHelper，需要在 Application.onCreate 时尽量早的地方调用：

```
SceneHelper.initialize(this);
```

该函数会把该场景 attach 到系统中，以 “告知” 系统场景进程已经启动，并参与到模块通讯机制中。

场景回调函数注册

为了能够接收指令消息（AICommand），需要（建议也在 Application.onCreate() 中）注册场景回调函数（包括生命周期函数）：

```
SceneHelper.setEventListener(new SceneEventListener() {

    @Override
    public void onSwitchIn(int flags) {
        super.onSwitchIn(flags);
```

```
    }

    /**
     * onCommand：进入场景(onSwitchIn)后，只需在onCommand中接收消息进行一系列
操作就可以了；
     * @param action 自定义行为
     * @param params 额外参数传递
     * @param suggestion 额外参数传递
     */
    @Override
    public void onCommand(String action, Bundle params, Serializable su
ggestion) {
        super.onCommand(action, params, suggestion);
    }
    @Override
    public void onSwitchOut() {
        super.onSwitchOut();
    }
});
```

其中 onCommand 函数的参数如注释所示。

实例说明

对于 Translator 场景，当用户发出类似 “我要翻译” 的语义指令后，系统将会收到类似以下的 AI 返回结果：

```
{
    "status": {
        "code": 0,
        "errorType": "success"
    },
    "query": "我要翻译",
    "semantic": {
        "service": "RTTranslator",
        "action": "Entry"
    }
}
```

"service": "RTTranslator" 即为ROS.AI系统里配置的场景，与布丁豆主控app里AndroidManifest.xml配置的模块名相对应。"action": "Entry" 即意图是"Entry"。依次顺序执行：

1. 调度启动 **RTTranslator** 场景里声明的 com.roobo.core.communication.DefaultService
2. 执行回调 onSwitchIn；
3. 执行回调 onCommand，其中参数中 action 为 Entry，params 和 suggestion 为

null;

场景接入操作完毕

布丁豆常用功能的使用

获取使用资源的能力

1) 以下代码代表获取使用camera的能力，只有返回的boolean值为真才可以使用camera

```
boolean reqCameraSucc = FocusManager.getInstance(MainActivity.this).req
uestFocus("camera");
```

2) 监听资源Focus状态

```
FocusManager.getInstance(MainActivity.this)
.addOnFocusChangedListener("camera", getModuleName(), new
OnModuleFocusChangedListener() {
    @Override
    public void onFocusWin(String focusName, String moduleName) {
        Log.i(TAG, "camera focus win");
    }
    @Override
    public void onFocusLost(String focusName, String moduleName) {
        // 释放camera对象
        Log.i(TAG, "camera focus lost");
    }
});
```

3) 释放资源
可在场景的Activity的onPause方法中释放资源

```
FocusManager.getInstance(MainActivity.this).releaseFocus("camera");
```

控制布丁豆转动

对布丁豆设备的操作需接入**robot.jar**
1） 获取控制设备的对象

```
// 获取控制设备旋转的对象
mMotionCtrlManager = MotionCtrlManager.getInstance();
mMotionCtrlManager.connect();
// 设置设备旋转的回调 MotionCtrlManager.RobotMotionStateListener
mMotionCtrlManager.setRobotMotionStateListener(this);
```

2) 旋转操作，以下有几种方式的旋转，可多次调用

```
mMotionCtrlManager.turnLeft(60, 10);      // 转到左侧60度位置
mMotionCtrlManager.reset(10);             // 归位 居中
mMotionCtrlManager.turnLeftToEnd(10);     // 左转到头
mMotionCtrlManager.turnRightToEnd(10);    // 右转到头
mMotionCtrlManager.getCurrentPosition(); // 获取当前位置
```

3) 断开连接

```
mMotionCtrlManager.disconnect();
```

TTS

1) 播放TTS并添加回调

```
PersonaPlayer.playTTS(this, "welcome china", new
PersonaPlayer.OnPlayerStateChangeListener() {
    @Override
    public void onActionStarted() {
        Log.i(TAG, "start play tts");
    }
    @Override
    public void onActionFinished() {
        Log.i(TAG, "finished play tts");
    }
});
```

2) 打断当前正在说的TTS

```
PersonaPlayer.playAction(this, "reset", null, null, null);
```