掌握焦点(Focus)和机器人生命周期

理解焦点 focus

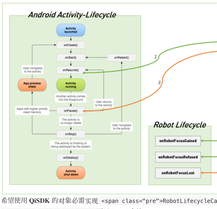
机器人焦点是 `Activity` 运行actions所必需的状态

它保证 `Activity` 一次只能拥有一个： `Activity` 被称作是机器人焦点的所有者.这确保机器人焦点所有者唯一控制机器人

机器人焦点由一项服务(service)管理，该服务用来将焦点给到activities. 下这种机制意味着 `Activity` 任何时候都可以获得或失去机器人的焦点.

机器人生命周期

QiSDK 提供了一个机器人的生命周期，允许任何 <activity> 拥有机器人的焦点.



希望使用 **QiSDK** 得对象必需实现 <RobotLifecycleCallbacks> 的接口. 比如 <Activity> 可以实现这个接口:

```
public class MyActivity extends RobotActivity implements RobotLifecycleCallbacks
```

此外，必须将此对象注册到 <Activity> 以接受到 <RobotLifecycleCallbacks> 方法
它必须在 <onCreate> 方法中注册:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Register the RobotLifecycleCallbacks to this Activity.
    QiSDK.register(this, this);
}
```

以及，在 <onDestroy> 方法中取消注册:

```
@Override
protected void onDestroy() {
    // Unregister the RobotLifecycleCallbacks for this Activity.
    QiSDK.unregister(this, this);
    super.onDestroy();
}
```

获得机器人焦点

当 `Activity` 获得 Android的焦点（focus）(移动到前台), 它会请求机器人的焦点.

当 `Activity` 获得机器人焦点，`onRobotFocusGained` 回调函数将会被注册到 `Activity` 上的每个 `RobotLifecycleCallbacks` 调用:

```
void onRobotFocusGained(QiContext qiContext);
```

Warning

此回调在 **后台线程**中运行, 所以你无法直接在其中操作UI组件.

它提供了一个 `QiContext` 允许你:

- 创建 actions.
- 为actions创建资源.
- 获取机器人的服务 （services）.

创建 **action**:

```
// Create a Say action with a QiContext.
Say say = SayBuilder.with(qiContext)
                    .withText("Hello")
                    .build();
```

创建依赖 **resource**:

```
// Create an Animation with a QiContext.
Animation animation = AnimationBuilder.with(qiContext)
                                      .withResources(R.raw.elephant_a001)
                                      .build();
```

获取服务 **service**:

```
// Get the HumanAwareness service with a QiContext.
HumanAwareness humanAwareness = qiContext.getHumanAwareness();
```

在这个回调中,你可以运行你的 actions:

运行 **action**:

```
// 运行一个同步的action.
say.run();
```

```
// 运行一个异步的action.
say.async().run();
```

```
<div class="section" id="losing-the-robot-focus">
  <h3>失去机器人焦点<a class="headerlink" href="#losing-the-robot-focus" title="Permalink to this headline"></a
  <p>当 <pre>Activity</pre> 失去机器人焦点， <pre>onRobotFocusLost</pre> 回调被每个注册在 <pre cla
```

```
void onRobotFocusLost();
```

```
    <div class="admonition warning">
```

对于actions的影响

调用此回调时，`Activity` 在重新获得机器人焦点之前，不能在Pepper上运行actions:

```
// This will fail if robot focus is lost.
say.run();
```

此外，如果 `onRobotFocusLost` 在操作正在运行时调用，则操作执行将停止并将引发异常:

```
// This will raise an exception.
say.run();
```

```
say.async().run().thenConsume(future -> {
    if (future.isSuccess()) {
        // This will not be called.
    } else if (future.hasError()) {
        // This will be called.
    }
});
```

因为Listeners可以在没有机器人焦点的情况下被触发，所以应当删除 `onRobotFocusLost` 回调下所有的Listeners:

```
// Remove listeners from LookAt.
if (lookAt != null) {
    lookAt.removeAllOnPolicyChangedListeners();
}
```

```
  <div class="section" id="impact-on-services">
    <h4>对服务 services的影响<a class="headerlink" href="#impact-on-services" title="Permalink to this headl
    </h4>
    <p>当机器人焦点丢失时，机器人服务不会受到影响.</p>
    <p>注意以异步services创建的 <pre class="pre">Future</pre> 将继续执行，并且仍将触发与任何服务关联的Listeners :
    </p>
```

```
// The Future will continue its execution.
Future<List<Human>> humansAroundFuture = humanAwareness.async().getHumansAround();
```

```
// The listener will still be triggered.
humanAwareness.addOnHumansAroundChangedListener(listener);
```

```
    <p>因为可以在没有机器人焦点的情况下触发Listeners，你应该删除所有在 <pre>onRobotFocusLost</pre> 回调中的
    listeners:</p>
    <div class="highlight-java notranslate">
      <div class="highlight">
        <pre><span></span><span class="c1">// Remove listeners from HumanAwareness.</span>
```

if (humanAwareness != null) { humanAwareness.removeAllOnHumansAroundChangedListeners(); }

如果一个 service listener 没有在 `onRobotFocusLost` 回调中移除, 则会在所有应用程序生命周期（lifecycle)内触发该Listener.

拒绝机器人焦点

在某些情况下，焦点可以拒绝你的 <Activity> ，例如机器人处于不稳定状态. <pre>onRobotFocusRefused</pre> 回调将会被每一个注册到 <Activity> 上的 <RobotLifecycleCallbacks> 调用:

```
void onRobotFocusRefused(String reason);
```

拒绝焦点的原因是作为参数提供的.

性能和局限

无法保证 `onRobotFocusGained` 或 `onRobotFocusRefused` 会被调用. 在某些情况下，`RobotLifecycleCallbacks` 可能永远不会得到获得焦点或焦点拒绝的通知.应用程序应该自己实现时间结束机制，以便处理这种意外情况并避免无限期等待.

```
<div class="section" id="get-back-on-ui-thread">
```

返回UI线程

您可能希望在机器人生命周期回调（robot lifecycle callbacks）中更新UI. 例如，你想更新 `TextView` .

要实现这一目标，您有两种常见的可能性.

基于Android的方案

`Activity` 类提供 `runOnUiThread` 方法运行一个 `Runnable` 在UI线程中:

```
@Override
public void onRobotFocusGained(QiContext qiContext) {

  Say say = SayBuilder.with(qiContext)
                      .withText("Hello")
                      .build();


  // Synchronous call.
    say.run();
    // Update the TextView to notify that the Say action is done.
    runOnUiThread(() -> textView.setText("Done!"));
}
```