处理运行时变更

本文内容

- 在配置变更期间保留对象
- 自行处理配置变更

另请参阅

- 提供资源
- 访问资源
- 加快屏幕方向变更
- 多窗口生命周期

有些设备配置可能会在运行时发生变化（例如屏幕方向、键盘可用性及语言）。 发生这种变化时，Android 会重启正在运行的 `Activity`（先后调用 `onDestroy()` 和 `onCreate()`）。重启行为旨在通过利用与新设备配置匹配的备用资源自动重新加载您的应用，来帮助它适应新配置。

要妥善处理重启行为，Activity 必须通过常规的Activity 生命周期恢复其以前的状态，在 Activity 生命周期中，Android 会在销毁 Activity 之前调用 `onSaveInstanceState()`，以便您保存有关应用状态的数据。 然后，您可以在 `onCreate()` 或 `onRestoreInstanceState()` 期间恢复 Activity 状态。

要测试应用能否在保持应用状态完好的情况下自行重启，您应该在应用中执行各种任务时调用配置变更（例如，更改屏幕方向）。 您的应用应该能够在不丢失用户数据或状态的情况下随时重启，以便处理如下事件：配置发生变化，或者用户收到来电并在应用进程被销毁很久之后返回到应用。 要了解如何恢复 Activity 状态，请阅读 Activity 生命周期。

但是，您可能会遇到这种情况：重启应用并恢复大量数据不仅成本高昂，而且给用户留下糟糕的使用体验。 在这种情况下，您有两个其他选择：

a. 在配置变更期间保留对象

   允许 Activity 在配置变更时重启，但是要将有状态对象传递给 Activity 的新实例。

b. 自行处理配置变更

   阻止系统在某些配置变更期间重启 Activity，但要在配置确实发生变化时接收回调，这样，您就能够根据需要手动更新 Activity。

在配置变更期间保留对象

如果重启 Activity 需要恢复大量数据、重新建立网络连接或执行其他密集操作，那么因配置变更而引起的完全重启可能会给用户留下应用运行缓慢的体验。 此外，依靠系统通过`onSaveInstanceState()` 回调为您保存的 `Bundle`，可能无法完全恢复 Activity 状态，因为它并非设计用于携带大型对象（例如位图），而且其中的数据必须先序列化，再进行反序列化，这可能会消耗大量内存并使得配置变更速度缓慢。 在这种情况下，如果 Activity 因配置变更而重启，则可通过保留 `Fragment` 来减轻重新初始化 Activity 的负担。此片段可能包含对您要保留的有状态对象的引用。

当 Android 系统因配置变更而关闭 Activity 时，不会销毁您已标记为要保留的 Activity 的片段。 您可以将此类片段添加到 Activity 以保留有状态的对象。

要在运行时配置变更期间将有状态的对象保留在片段中，请执行以下操作：

1. 扩展 `Fragment` 类并声明对有状态对象的引用。
2. 在创建片段后调用 `setRetainInstance(boolean)`。
3. 将片段添加到 Activity。

4. 重后 Activity 后，使用 `FragmentManager` 检索片段。

例如，按如下方式定义片段：

```
public class RetainedFragment extends Fragment {

    // data object we want to retain
    private MyDataObject data;

    // this method is only called once for this fragment
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // retain this fragment
        setRetainInstance(true);
    }

    public void setData(MyDataObject data) {
        this.data = data;
    }

    public MyDataObject getData() {
        return data;
    }
}
```

> **注意：**尽管您可以存储任何对象，但是切勿传递与 `Activity` 绑定的对象，例如，`Drawable`、`Adapter`、`View` 或其他任何与 `Context` 关联的对象。否则，它将泄漏原始 Activity 实例的所有视图和资源。（泄漏资源意味着应用将继续持有这些资源，但是无法对其进行垃圾回收，因此可能会丢失大量内存。）

然后，使用 `FragmentManager` 将片段添加到 Activity。在运行时配置变更期间再次启动 Activity 时，您可以获得片段中的数据对象。例如，按如下方式定义 Activity：

```
public class MyActivity extends Activity {

    private RetainedFragment dataFragment;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // find the retained fragment on activity restarts
        FragmentManager fm = getFragmentManager();
        dataFragment = (DataFragment) fm.findFragmentByTag(“data”);

        // create the fragment and data the first time
        if (dataFragment == null) {
            // add the fragment
            dataFragment = new DataFragment();
            fm.beginTransaction().add(dataFragment, “data”).commit();
            // load the data from the web
            dataFragment.setData(loadMyData());
        }

        // the data is available in dataFragment.getData()
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // store the data in the fragment
        dataFragment.setData(collectMyLoadedData());
    }
}
```

在此示例中，`onCreate()` 添加了一个片段或恢复了对它的引用。此外，`onCreate()` 还将有状态的对象存储在片段实例内部。`onDestroy()`

对所保留的片段实例内的有状态对象进行更新。

自行处理配置变更

如果应用在特定配置变更期间无需更新资源，并且因性能限制您需要尽量避免重启，则可声明 Activity 将自行处理配置变更，这样可以阻止系统重启 Activity。

> **注：**自行处理配置变更可能导致备用资源的使用更为困难，因为系统不会为您自动应用这些资源。 只能在您必须避免 Activity 因配置变更而重启这一万般无奈的情况下，才考虑采用自行处理配置变更这种方法，而且对于大多数应用并不建议使用此方法。

要声明由 Activity 处理配置变更，请在清单文件中编辑相应的 `<activity>` 元素，以包含 `android:configChanges` 属性以及代表要处理的配置的值。`android:configChanges` 属性的文档中列出了该属性的可能值（最常用的值包括 `"orientation"` 和 `"keyboardHidden"`，分别用于避免因屏幕方向和可用键盘改变而导致重启）。您可以在该属性中声明多个配置值，方法是用管道 `|` 字符分隔这些配置值。

例如，以下清单文件代码声明的 Activity 可同时处理屏幕方向变更和键盘可用性变更：

```
<activity android:name=".MyActivity"
          android:configChanges="orientation|keyboardHidden"
          android:label="@string/app_name">
```

现在，当其中一个配置发生变化时，`MyActivity` 不会重启。相反，`MyActivity` 会收到对 `onConfigurationChanged()` 的调用。向此方法传递 `Configuration` 对象指定新设备配置。您可以通过读取 `Configuration` 中的字段，确定新配置，然后通过更新界面中使用的资源进行适当的更改。调用此方法时，Activity 的 `Resources` 对象会相应地进行更新，以根据新配置返回资源，这样，您就能够在系统不重启 Activity 的情况下轻松重置 UI 的元素。

> **注意：**从 Android 3.2（API 级别 13）开始，当设备在纵向和横向之间切换时，“**屏幕尺寸**”**也会发生变化**。因此，在开发针对 API 级别 13 或更高版本（正如 `minSdkVersion` 和 `targetSdkVersion` 属性中所声明）的应用时，若要避免由于设备方向改变而导致运行时重启，则除了 `"orientation"` 值以外，您还必须添加 `"screenSize"` 值。 也就是说，您必须声明 `android:configChanges="orientation|screenSize"`。但是，如果您的应用面向 API 级别 12 或更低版本，则 Activity 始终会自行处理此配置变更（即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重启 Activity）。

例如，以下 `onConfigurationChanged()` 实现检查当前设备方向：

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Checks the orientation of the screen
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
}
```

`Configuration` 对象代表所有当前配置，而不仅仅是已经变更的配置。大多数时候，您并不在意配置具体发生了哪些变更，而且您可以轻松地重新分配所有资源，为您正在处理的配置提供备用资源。 例如，由于 `Resources` 对象现已更新，因此您可以通过 `setImageResource()` 重置任何 `ImageView`，并且使用适合于新配置的资源（如提供资源中所述）。

请注意，`Configuration` 字段中的值是与 `Configuration` 类中的特定常量匹配的整型数。有关要对每个字段使用哪些常量的文档，请参阅 `Configuration` 参考文档中的相应字段。

> **请谨记：**在声明由 Activity 处理配置变更时，您有责任重置要为其提供备用资源的所有元素。 如果您声明由 Activity 处理方向变更，而且有些图像应该在横向和纵向之间切换，则必须在 `onConfigurationChanged()` 期间将每个资源重新分配给每个元素。

如果无需基于这些配置变更更新应用，则可*不用*实现 `onConfigurationChanged()`。在这种情况下，仍将使用在配置变更之前用到的所有资源，只是您无需重启 Activity。 但是，应用应该始终能够在保持之前状态完好的情况下关闭和重启，因此您不得试图通过此方法来逃避在正常 Activity 生命周期期间保持您的应用状态。 这不仅仅是因为还存在其他一些无法禁止重启应用的配置变更，还因为有些事件必须由您处理，例如用户离开应用，而在用户返回应用之前该应用已被销毁。

如需了解有关您可以在 Activity 中处理哪些配置变更的详细信息，请参阅 `android:configChanges` 文档和 `Configuration` 类。