字符串资源

字符串资源为您的应用提供具有可选文本样式和格式设置的文本字符串。 共有三种类型的资源可为您的应用提供字符串：

String

提供单个字符串的 XML 资源。

String Array

提供字符串数组的 XML 资源。

Quantity Strings (Plurals)

带有用于多元化的不同字符串的 XML 资源。

所有字符串都能应用某些样式设置标记和格式设置参数。如需了解有关样式和格式设置字符串的信息，请参阅有关格式和样式设置的部分。

String

可从应用或从其他资源文件（如 XML 布局）引用的单个字符串。

> **注**：字符串是一种使用 `name` 属性（并非 XML 文件的名称）中提供的值进行引用的简单资源。 因此，您可以在一个 XML 文件中将字符串资源与其他简单资源合并在一起，放在 `<resources>` 元素之下。

文件位置：

`res/values/filename.xml`
filename 是任意值。`<string>` 元素的 `name` 将用作资源 ID。

编译的资源数据类型：

指向 `String` 的资源指针。

资源引用：

在 Java 中：`R.string.string_name`
在 XML 中：`@string/string_name`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string
        name="string_name"
        >text_string</string>
</resources>
```

元素：

`<resources>`

必备。此元素必须是根节点。

无属性。

`<string>`

一个字符串，可包括样式设置标记。请注意，您必须将撇号和引号转义。 如需了解有关如何正确设置字符串样式和格式的详细信息，请参阅下文的格式和样式设置。

属性：

`name`

String。字符串的名称。该名称将用作资源 ID。

示例：

保存在 `res/values/strings.xml` 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

该布局 XML 会对视图应用一个字符串：

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

以下应用代码用于检索字符串：

```
String string = getString(R.string.hello);
```

您可以使用 `getString(int)` 或 `getText(int)` 来检索字符串。`getText(int)` 将保留应用于字符串的任何富文本样式设置。

String Array

可从应用引用的字符串数组。

> **注**：字符串数组是一种使用 `name` 属性（并非 XML 文件的名称）中提供的值进行引用的简单资源。 因此，您可以在一个 XML 文件中将字符串数组资源与其他简单资源合并在一起，放在 `<resources>` 元素之下。

文件位置：

`res/values/`*`filename`*`.xml`
filename 是任意值。`<string-array>` 元素的 `name` 将用作资源 ID。

编译的资源数据类型：

指向 `String` 数组的资源指针。

资源引用：

在 Java 中：`R.array.`*`string_array_name`*

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="string_array_name">
        <item
            >text_string</item>
    </string-array>
</resources>
```

元素：

`<resources>`

必备。此元素必须是根节点。

无属性。

`<string-array>`

定义一个字符串数组。包含一个或多个 `<item>` 元素。

属性：

`name`

String。数组的名称。该名称将用作资源 ID 来引用数组。

`<item>`

一个字符串，可包括样式设置标记。其值可以是对另一字符串资源的引用。 必须是 `<string-array>` 元素的子项。请注意，您必须将撇号和引号转义。 如需了解有关如何正确设置字符串样式和格式的信息，请参阅下文的格式和样式设置。

无属性。

示例：

保存在 `res/values/strings.xml` 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

以下应用代码用于检索字符串数组：

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Quantity Strings (Plurals)

不同语言在语法数量一致上具有不同的规则。例如，在英语中，数量 1 是一种特殊情况。 我们会写成“1 book”，但如果是任何其他数量，我们则会写成“*n* books”。 这种对单复数的区分很常见，但其他语言进行了更加细致的区分。 Android 支持的完整集合包括 `zero`、`one`、`two`、`few`、`many` 和 `other`。

决定为给定语言和数量使用哪一种情况的规则可能非常复杂，因此 Android 为您提供了 `getQuantityString()` 等方法来选择适合您的资源。

尽管历史上被称作“数量字符串”（并且在 API 中也仍然这样叫），但数量字符串 *只*应用于表示复数。 例如，使用数量字符串来实现 Gmail

的“Inbox”之类的情况是错误的，正确的做法是使用它们来实现“Inbox (12)”这种存在未读邮件的情况。 使用数量字符串来替代 `if` 语句似乎更为方便，但必须注意的是，某些语言（如中文）根本不做这些语法区分，因此您获取的始终是 `other` 字符串。

选择使用哪一个字符串完全取决于语法上的*必要性*。在英语中，即使数量是 0，一个表示 `zero` 的字符串也会被忽略，因为在语法上 0 与 2 或 1 以外的任何其他数字没有区别（“zero books”、“one book”、“two books”、等等）。 相反，在韩语中，得到使用的就*只有* `other` 字符串。

不要被某些事实误导，比如 `two` 听起来只能应用于数量 2：某种语言可能规定，2、12、102（等等）均相同对待，但与其他数量则区分对待。可以依靠翻译人员来了解他们的语言实际的区分要求。

通常可以利用“Books: 1”之类的数量中性表示来避免使用数量字符串。 如果这是一种符合您的应用需要的样式，就能减轻您和翻译人员的工作负荷。

> **注**：Plurals 集合是一种使用 `name` 属性（并非 XML 文件的名称）中提供的值进行引用的简单资源。 因此，您可以在一个 XML 文件中将 plurals 资源与其他简单资源合并在一起，放在 `<resources>` 元素之下。

文件位置：

`res/values/`*`filename`*`.xml`
filename 是任意值。`<plurals>` 元素的 `name` 将用作资源 ID。

资源引用：

在 Java 中：`R.plurals.`*`plural_name`*

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals
        name="plural_name">
        <item
            quantity=["zero" | "one" | "two" | "few" | "many" | "other"]
            >text_string</item>
    </plurals>
</resources>
```

元素：

`<resources>`

必备。此元素必须是根节点。

无属性。

`<plurals>`

一个字符串集合，根据事物数量提供其中的一个字符串。 包含一个或多个 `<item>` 元素。

属性：

`name`

String。字符串对的名称。该名称将用作资源 ID。

`<item>`

一个复数或单数字符串。其值可以是对另一字符串资源的引用。 必须是 `<plurals>` 元素的子项。请注意，您必须将撇号和引号转义。 如需了解有关如何正确设置字符串样式和格式的信息，请参阅下文的格式和样式设置。

属性：

`quantity`

关键字。表示应在何时使用该字符串的值。以下是其有效值，括号内的示例并不详尽：

值	说明
`zero`	当语言要求对数字 0 做特殊对待时（如阿拉伯语的要求）。
`one`	当语言要求对 1 这类数字做特殊对待时（如英语和大多数其他语言中对数字 1 的对待要求；在俄语中，任何末尾是 1 但不是 11 的数字均属此类）。
`two`	当语言要求对 2 这类数字做特殊对待时（如威尔士语中对 2 的要求，或斯洛文尼亚语中对 102 的要求）。
`few`	当语言要求对“小”数字做特殊对待时（如捷克语中的 2、3 和 4；或波兰语中末尾是 2、3 或 4 但不是 12、13 或 14 的数字）。
`many`	当语言要求对“大”数字做特殊对待时（如马耳他语中末尾是 11-99 的数字）。
`other`	当语言不要求对给定数量做特殊对待时（如中文中的所有数字，或英语中的 42）。

示例：

保存在 `res/values/strings.xml` 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <!--
             As a developer, you should always supply "one" and "other"
             strings. Your translators will know which strings are actually
             needed for their language. Always include %d in "one" because
             translators will need to use %d for languages where "one"
             doesn't mean 1 (as explained above).
          -->
        <item quantity="one">%d song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>
```

保存在 `res/values-pl/strings.xml` 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">Znaleziono %d piosenkę.</item>
        <item quantity="few">Znaleziono %d piosenki.</item>
        <item quantity="other">Znaleziono %d piosenek.</item>
    </plurals>
</resources>
```

Java 代码：

```
int count = getNumberOfsongsAvailable();
Resources res = getResources();
String songsFound = res.getQuantityString(R.plurals.numberOfSongsAvailable, count, count);
```

使用 `getQuantityString()` 方法时，如果您的字符串包括的字符串格式设置带有数字，则需要传递 `count` 两次。 例如，对于字符串 `%d songs found`，第一个 `count` 参数选择相应的复数字符串，第二个 `count` 参数将插入 `%d` 占位符内。 如果您的复数字符串不包括字符串格式设置，则无需向 `getQuantityString` 传递第三个参数。

格式和样式设置

关于如何正确设置字符串资源的格式和样式，您应该了解下面这几个要点。

转义撇号和引号

如果字符串中包含撇号 (')，您必须用反斜杠 (\') 将其转义，或为字符串加上双引号 ("")。 例如，以下是一些有效和无效的字符串：

```
<string name="good_example">This\'ll work</string>
<string name="good_example_2">"This'll also work"</string>
<string name="bad_example">This doesn't work</string>
    <!-- Causes a compile error -->
```

如果字符串中包含双引号，您必须将其转义（使用 \"）。 为字符串加上单引号不起作用。

```
<string name="good_example">This is a \"good string\".</string>
<string name="bad_example">This is a "bad string".</string>
    <!-- Quotes are stripped; displays as: This is a bad string. -->
<string name="bad_example_2">'This is another "bad string".'</string>
    <!-- Causes a compile error -->
```

设置字符串格式

如果您需要使用 `String.format(String, Object...)` 设置字符串格式，可以通过在字符串资源中加入格式参数来实现。 例如，对于以下资源：

```
<string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

在本例中，格式字符串有两个参数：`%1$s` 是一个字符串，而 `%2$d` 是一个十进制数字。 您可以像下面这样使用应用中的参数设置字符串格式：

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
```

使用 HTML 标记设置样式

您可以使用 HTML 标记为字符串添加样式设置。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

支持的 HTML 元素包括：

- `<b>` 表示**粗体**文本。
- `<i>` 表示*斜体*文本。
- `<u>` 表示 下划线 文本。

有时，您可能想让自己创建的带样式文本资源同时也用作格式字符串。 正常情况下，这是行不通的，因为 `String.format(String, Object...)` 方法会去除字符串中的所有样式信息。 这个问题的解决方法是编写带转义实体的 HTML 标记，在完成格式设置后，这些实体可通过 `fromHtml(String)` 恢复。 例如：

1. 将您带样式的文本资源存储为 HTML 转义字符串：

   ```
   <resources>
     <string name="welcome_messages">Hello, %1$s! You have &lt;b>%2$d new messages&lt;/b>.</string>
   </resources>
   ```

   在这个带格式的字符串中，添加了 `<b>` 元素。请注意，开括号使用 `<` 表示法进行了 HTML 转义。

2. 然后照常设置字符串格式，但还要调用 `fromHtml(String)` 以将 HTML 文本转换成带样式文本：

   ```
   Resources res = getResources();
   String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
   CharSequence styledText = Html.fromHtml(text);
   ```

由于 `fromHtml(String)` 方法将设置所有 HTML 实体的格式，因此务必要使用 `htmlEncode(String)` 对您用于带格式文本的字符串中任何可能的 HTML 字符进行转义。 例如，如果您向 `String.format()` 传递的字符串参数可能包含“<”或“&”之类的字符，则必须在设置格式前进行转义，这样在通过 `fromHtml(String)` 传递带格式字符串时，字符就能以原始形式显示出来。 例如：

```
String escapedUsername = TextUtil.htmlEncode(username);

Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), escapedUsername, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

使用 Spannable 设置样式

`Spannable` 是一种文本对象，让您可以使用颜色和字体粗细等字体属性进行样式设置。 您可以使用 `SpannableStringBuilder` 生成文本，然后对文本应用 `android.text.style` 包中定义的样式。

您可以利用下列辅助工具方法来设置许多 spannable 文本创建工作：

```
/**
 * Returns a CharSequence that concatenates the specified array of CharSequence
 * objects and then applies a list of zero or more tags to the entire range.
 *
 * @param content an array of character sequences to apply a style to
 * @param tags the styled span objects to apply to the content
 *        such as android.text.style.StyleSpan
 *
 */
private static CharSequence apply(CharSequence[] content, Object... tags) {
    SpannableStringBuilder text = new SpannableStringBuilder();
    openTags(text, tags);
    for (CharSequence item : content) {
        text.append(item);
    }
    closeTags(text, tags);
    return text;
}

/**
 * Iterates over an array of tags and applies them to the beginning of the specified
 * Spannable object so that future text appended to the text will have the styling
 * applied to it. Do not call this method directly.
 */
private static void openTags(Spannable text, Object[] tags) {
    for (Object tag : tags) {
        text.setSpan(tag, 0, 0, Spannable.SPAN_MARK_MARK);
    }
}

/**
 * "Closes" the specified tags on a Spannable by updating the spans to be
 * endpoint-exclusive so that future text appended to the end will not take
 * on the same styling. Do not call this method directly.
 */
private static void closeTags(Spannable text, Object[] tags) {
    int len = text.length();
    for (Object tag : tags) {
        if (len > 0) {
            text.setSpan(tag, 0, len, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
        } else {
            text.removeSpan(tag);
        }
    }
}
```

以下 `bold`、`italic` 和 `color` 方法向您展示了如何调用这些帮助程序方法来应用 `android.text.style` 包中定义的样式。 您可以创建类似方法来进行其他类型的文本样式设置。

```
/**
 * Returns a CharSequence that applies boldface to the concatenation
 * of the specified CharSequence objects.
 */
public static CharSequence bold(CharSequence... content) {
    return apply(content, new StyleSpan(Typeface.BOLD));
}

/**
 * Returns a CharSequence that applies italics to the concatenation
 * of the specified CharSequence objects.
 */
public static CharSequence italic(CharSequence... content) {
    return apply(content, new StyleSpan(Typeface.ITALIC));
}

/**
 * Returns a CharSequence that applies a foreground color to the
 * concatenation of the specified CharSequence objects.
 */
public static CharSequence color(int color, CharSequence... content) {
    return apply(content, new ForegroundColorSpan(color));
}
```

下面这个示例展示了如何将这些方法链接起来，创建出对不同词语应用不同类型样式的字符序列：

```
// Create an italic "hello, " a red "world",
// and bold the entire sequence.
CharSequence text = bold(italic(res.getString(R.string.hello)),
    color(Color.RED, res.getString(R.string.world)));
```