双向文本元素在 SVG 中的显示技术①

刘 旭

(SAP 中国研究院 商务智能部, 上海 201203)

摘 要: 双向文本在SVG中的显示是使用Unicode双向算法实现的, 然而在很多情况下双向算法无法自动生成符合语义的正确显示结果, 需要进行额外的设置. 通过分析双向算法的基本原理, 深入探讨了双向文本显示中容易出现问题的几个方面, 包括文本方向的判定, 特殊字符显示位置, 文本元素中跨越tspan元素的文字重排, 以及设置显示方向之后位置的变化. 针对各个问题的具体情况和不同浏览器的差异, 使用 SVG 中的特定属性给出了进行正确设置的方式.

关键词: 双向算法; SVG; 从右到左; Unicode; 可视化

Displaying Technology of Bidirectional Text Element in SVG

LIU Xu

(Department of Business Intelligence of SAP Labs China, Shanghai 201203, China)

Abstract: The displaying of bidirectional text in SVG is implemented by Unicode bidirectional algorithm, but bidirectional algorithm cannot automatically generate correct results to be displayed in line with semantics in many cases, so additional settings are required. By analyzing the basic principle of bidirectional algorithm, this paper discusses several aspects of bidirectional text display prone to occur, which includes text direction determining, display positions of special characters, text reordering across tspan elements in text elements, and position changes after display direction setting. According to specific circumstances of each issue and individual differences in different browsers, figure out ways to set correctly by using specific attributes in SVG.

Key words: bidirectional algorithm; SVG; right-to-left; Unicode; visualization

中文和英文都是横向从左到右(Left-To-Right, 简称 LTR)排布的文字, 然而, 世界上还有大量的人使用横向从右到左(Right-To-Left, 简称 RTL)排布的文字, 阿拉伯语, 波斯语, 希伯来语,以及中国的维吾尔语都是这样的语言[1]. 在国际化软件的开发过程中, 不可避免地会涉及到对于从左到右与从右到左文本混排的显示, 例如维吾尔语和汉语的混排, 或者英语和阿拉伯语的混排[2]. 不同于传统的从左到右显示的汉语或英语, 双向文本中由于存在从右到左字符, 其显示带来可能一些特殊的问题, 如果不能正确处理, 就会造成显示上的混乱, 让文本阅读者产生误解.

在可视化软件开发中, SVG 是 HTML5 中常用的矢量图形技术[3], JavaScript 语言兼具面向对象和函数式编程风格[4], 使用JavaScript结合SVG已经在可视化软件开发中得到广泛应用, 包括各类科学可视化[5]和信息可视化软件[6], 而在开发中往往使用 text 元素(Text Element)为 SVG 中的图形标示文本标签[7]. 随着国际化软件的增多, 没有使用正确的显示技术而带来的 text 元素中的双向文字问题也越来越常见.

1 双向文字方向的确定

如果使用"…"代表要显示的文字, 在SVG中设置文字元素的代码一般为

```
<text>…</text>
```

双向文字在 text 元素中显示首先要解决的问题, 就是浏览器必须知道字符串中的哪些字符应该从左到

① 收稿时间:2016-07-20;收到修改稿时间:2016-09-08 [doi: 10.15888/j.cnki.csa.005698]

右显示, 哪些字符应该从右到左显示. 有些情况下字符排布方向的判断很容易, 但是有些情况下是难以判断的. 例如英文与阿拉伯语混排的情况下, 阿拉伯语词组中的阿拉伯字母必须从右到左排列, 而英文词组中的字母必须从左到右排列, 各不同语种词组之间的排列方向确定就更加复杂.

为了解决双向文字排布的问题, 现代的浏览器都支持双向文字算法[8], 各种双向文字算法中最常用的是 Unicode 双向算法 (Unicode Bidirectional Algorithm)[9]. Unicode 双向算法的基本思路是将字符串中的字符按照对于排布方向的要求分为强(Strong)类型, 弱(Weak)类型, 中立(Neutral)类型等几类[10], 通过判断字符之间的相对位置[11], 将字符串分为若干个称为 run 的区段, 一个 run 包含一个或多个单词, 每一个 run 内部字符的方向是一致的, 各个 run 之间的方向则根据文本元素的基本方向来确定[12], 默认的基本方向是从左到右. 不同的浏览器实现的双向文字算法细节上可能不同, 若未经声明, 本文提到的浏览器是指 Google Chrome 浏览器 Version 50.

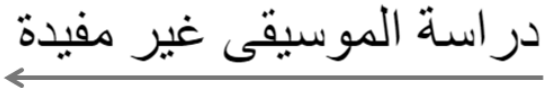

图 1 阿拉伯语的默认从右到左显示

在不附加任何特殊属性的情况下, 如果一句话完全由阿拉伯语字母构成, 此时整个文本是一个 run, 浏览器可以将其从右到左正常显示. 图 1 是一句阿拉伯语"学习音乐是有用的"的显示.一个简单的阿拉伯语和英语混排的双向字符串(含义为"学习 SVG 1.2 是有用的")在 SVG 的 text 元素中将显示为图 2 的样子, 可以看到这句话被分成了三个 run, (1)标示的阿拉伯语含义是"学习", 可见在 run 这个粒度上, 字符串的顺序是从左到右的. (1)和(3)在 run 的内部字符都是从右到左的, 而(2)的内部字符还是从左到右的.

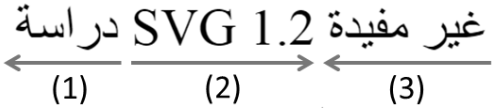

图 2 默认情况下双向文字的显示

虽然从字符的角度来看图 2 显示的是混合语言文本, 但是从语义的角度来看, 图 1 显示的显然是一句阿拉伯语, 只不过阿拉伯语中混入了英文词组. 参照图 1 的显示不难得知, 如图 2 那样的显示方式不符合阿拉伯语使用习惯, 因为"学习"这个词明显应该显示在最右边. 要从根本上解决这个问题, 需要浏览器的双向文字算法能够根据语义确定出这是一句阿拉伯语. 在文本量足够大的情况下[13], 例如在一些翻译软件中, 可以通过语法树解析或是文本 Unicode 编码统计的方法确定文本的语言[14], 从而判断出文本的方向, 但 SVG 文本标签中的文本往往只有很少的几个单词, 难以进行有效的语言检测.

SVG 中的 Unicode 双向算法引入的解决方案是让开发人员在 text 元素中使用 direction 属性设置文本方向, 从而提示 Unicode 双向算法应该使用什么样的方向排布已经划分好的 run, 设置方式形如

<text direction="rtl">...</text>

图 3 是将 direction 属性设为 rtl 之后的图 2 中的文字显示情况, 可以看到(2)还在原来的位置, 而(1)和(3)互换了位置. 参照图 1, 可以看到图 3 完全符合阿拉伯语的显示方式. 在 SVG 软件开发中, 由于开发者事先往往不能确定从右到左语言的字符串中没有任何从左到右字符, 为了正确显示从右到左语言, direction="rtl" 的设置是必不可少的.

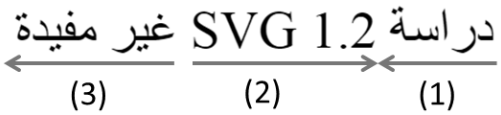

图 3 设置 direction="rtl"后双向文字的显示

在一些非常特殊的情况下, 可能需要保证所有的字符都严格从右到左显示, 这时候可以将文本元素的 unicode-bidi 属性设为 bidi-override, 设置方式形如

<text direction="rtl"
unicode-bidi="bidi-override">...</text>

这种设置方式其实就是将文本方向的判断作了简单化处理. 图 4 显示了图 2 中的文字在这种设置下的显示情况, 可以直观地看出这种显示方式技术上的实现很简单, 然而"SVG 1.2"的显示完全不符合英语的习惯, 难以读懂其含义. 正因为如此, 实际的 SVG 程序开发中很少将 unicode-bidi 属性设为 bidi-override, 这也从一个侧面反映出双向算法为字符串划分 run 的必要性.

دراسة GVS 2.1 غير مفيدة

图 4 使用 bidi-override 之后的双向字符串

2 特殊字符的方向控制

引入 run 的划分和文字方向的设定之后, 双向文本的显示问题集中在如何正确划分出文本的 run, 而这往往涉及对于文本语义的分析, 特别是存在特殊字符的情况下.

在现代实际使用的各种语言中, 大量存在空格和标点符号, 正负号, 百分号等特殊字符. 单个的特殊字符在从左到右和从右到左语言中的显示都相同, 然而在不同的上下文中, 它们有时应该从左到右显示, 有时应该从右到左显示. 在 Unicode 双向算法中往往将这些特殊字符划分为弱类型或中立类型, 如果不能正确判断特殊字符的方向, 就无法正确显示文本. 图 5 显示的字符串设置了 direction="rtl", 然而(1)处的句号显然应该显示在"The number is -12.3"的右边, 而(2)处试图显示的数字为-12.3, 即使在阿拉伯语中, 负号也应该显示在 12.3 的左边. 出现这种情况的原因是双向算法没能正确地判定某些特殊字符所属于的 run, 从而使符号的位置无法反映对应的语义, 两个同样的 -12.3, 在不同的位置显示就不同. 文本中常见的正负号, 句号, 百分号,省略号都可能有这种情况

The number is -12.3. (1) عدد غير -12.3 (2) في هذه المادة.

图 5 特殊字符无法正确显示的情况

为了解决这个问题, 最根本的解决方案是优化现有的双向算法, 使其可以正确判断出特殊字符的方向. 然而, 这在很多情况下无法简单地通过前后字符的编码来判断, 必须进行语法解析, 例如图 5 中的例子, 算法必须判断出(1)处的句号是属于英语句子. 即使这样, 不同的方言区, 不同的特殊字符都有不同的使用习惯, 例如对于负数, 标准阿拉伯语中习惯跟英语中一样将负号标注在左边, 如-10, 但是对于百分数, 标准阿拉伯语却习惯将百分号也标注在左边, 如%10, 跟英语恰好相反, 然而有的阿拉伯语使用地区, 如伊朗地区却仍然习惯将百分号标注在右边, 如 10%, 与英语相同. 如果双向算法要实现全部特殊字符方向的自动判断, 其设计必然相当复杂, 而且在文本很少的情况下判断的准确性仍然难以保证.

Unicode 双向算法使用的方法是允许算法的使用人员标注特殊字符的方向. Unicode 双向算法将按照这些标注结合上下文的字符进行 run 的划分. SVG 中提供的标注方法有两种, 一种是使用专门定义的不可见 Unicode 字符进行方向标注[15], 一种是使用内联元素 tspan 结合 unicode-bidi 属性设为 embed 来标注, W3C 组织推荐的方法是第二种. 对于图 5 中的情况, 可以将图 5(1)处的句号和图 5(2)处的负号置入 tspan 元素中, 方向标记为 ltr. 如果用...代替其他文本, 设置代码形如:

```
<text direction="rtl">...<tspan direction="ltr"
unicode-bidi="embed">.</tspan>...
<tspan direction="ltr" unicode-bidi="embed">-
</tspan>12.3...</text>
```

图 6 是这样设置之后的显示, 可以看到两处负号都显示在数字的左边,符合数学上的要求, 而两处句号分别根据每个句子中的文字方向来显示, 英文的句子末尾句号显示在整句话的最右边, 而阿拉伯语的句子末尾句号显示在文字的最左边, 符合两种语言的阅读习惯.

The number is -12.3. عدد غير 12.3- في هذه المادة.

图 6 使用 tspan 进行设置后特殊字符的正确显示

在双向字符串中的括号, 引号等符号也会出现类似问题. 括号, 引号等字符是成对出现的(称为 Mirrored characters, 镜像字符), 以括号为例, Unicode 双向算法有镜像替换功能, 将一处括号的文字方向判断之后, 会相应地改变括号的方向[16], 有时候会造成相当费解的显示错误. 图 7 上方的字符串是设置了 direction="rtl"之后的双向文字, 可见字符串中出现了两处左括号, 而画圈处的括号明显应该是一个显示在 Graphics 之后的右括号, 即结束括号. 出现这种情况的原因是 Unicode 双向算法将画圈处的括号判断为从右到左字符, 而在从右到左文本中, 结束括号应该显示为从左到右文本中左括号的样子.

دراسة (SVG (Scalable Vector Graphics غير مفيدة

دراسة (SVG (Scalable Vector Graphics غير مفيدة

图 7 镜像字符显示的设置

为了解决这个问题, 需要像图 6 那样对结束的括号使用内联元素 tspan, direction 设为 ltr, 并将 unicode-bidi 属性设为 embed. 图 7 下方的字符串即正确显示的结果. SVG 中完整的设置形如:

```
<text direction="rtl">…SVG (Scalable Vector Graphics
<tspan direction="ltr" unicode-bidi="embed">)
</tspan>…</text>
```

3 tspan元素的文字重排问题

双向文本在 SVG 中的显示还有一些与使用的元素相关的问题. 在前文的分析中多次提到的tspan元素是内联(inline)元素, 在 SVG 开发中, 经常使用 tspan 对某些文本进行格式的设置, 例如加粗体, 改变颜色, 改变位置等[17]. 然而, 内联元素不像块(block)元素一样有自己占据的固定空间, 它是依附于块元素存在的, 特别地, 在双向文本的情况下, 如果通过设置 direction="rtl"改变了文字方向, text 元素中的文本可能跨越tspan元素进行重排, 从而可能出现意料不到的结果. 以下是一段在 text 元素中放入两个 tspan 元素, 通过设置适当的 dy 属性值显示两行文字的 SVG 示例, 其中的省略号代表阿拉伯文字:

```
<text>
    <tspan dy="1em">Studying JavaScript 1.8 is
helpful</tspan>
    <tspan dy="1em">…JavaScript 1.8…</tspan>
</text>
```

Studying JavaScript 1.8 is helpful
دراسة JavaScript 1.8 غير مفيدة

图 8 使用 tspan 元素显示为两行的双向文本

显示效果如图 8 所示, 文本确实已经显示为两行, 然而第二行的显示存在图 2 提到的问题, 不符合阿拉伯语的使用习惯. 如果按照前文所述, 在 text 元素上设置 direction="rtl"来解决第二行的这个问题, 将 SVG 代码改为:

```
<text direction="rtl">
    <tspan dy="1em">Studying JavaScript 1.8 is
helpful</tspan>
    <tspan dy="1em">…JavaScript 1.8…</tspan>
</text>
```

那么会得到如图 9 所示的显示结果. 可以看到第一行和第二行的文本都发生了变化, 第一行原本英文显示的内容变成了阿拉伯语, 而原本在第一行的单词"Studying"甚至被分成了两个部分显示, 这显然不是我们期望的结果.

Stud دراسة JavaScript 1.8 غير مفيدة
ying JavaScript 1.8 is helpful

图 9 在 text 元素上设置 direction="rtl"之后的两行双向文本

出现图 9 这种显示结果的原因在于 Unicode 双向算法在处理 text 元素中字符串方向时, 可以跨 tspan 元素进行. 这意味着虽然我们将 text 元素中的字符串拆分在了两个 tspan 子元素里面, 在双向算法判断文本方向时并不考虑 tspan 的分割, 而是将所有文本当成一句话处理. 由于"Studying JavaScript 1.8 is helpful" 先于阿拉伯语句子显示, 在从右到左显示的情况下, 就应该从 text 元素内部的"右边", 即第二个 tspan 开始显示, 而第二个 tspan 的位置被设置为第二行, 于是整个文本就成了从第二行开始显示, 到第一行为止, 阿拉伯语句子显示在了第一行. 需要注意的是, 每一行文本 的宽度仍然是按照默认的从左到右情况下的文本长度确定的, 参照图 8 可以直观地看出. 由于第一行预留的宽度比阿拉伯语句子宽, 英文句子的一部分显示在了第一行, 从而出现了一个单词在上下两行显示的情况.

要正确显示图 8 的文本, 应该把 direction="rtl"设置在 tspan 元素上, 同时需要告诉 Unicode 双向算法在设置 tspan 中的文本方向时, 不要考虑全部的字符串. 这可以通过设置 unicode-bidi="isolate"来做到[18]. 代码可以修改为:

```
<text>
    <tspan dy="1em" unicode-bidi="isolate"
direction="rtl">Studying JavaScript 1.8 is helpful</tspan>
    <tspan dy="1em" unicode-bidi="isolate"
direction="rtl">…JavaScript 1.8…</tspan>
</text>
```

图 10 显示了设置 unicode-bidi="isolate"之后的两行双向文本, 参照图3, 可以看到第二行的显示符合阿拉伯语的使用习惯. 至于第一行的英文字符串, Unicode 双向算法能正确判断出其属于一个 run, 不受

direction="rtl"的影响, 然而, 这里加上 direction="rtl"的设置可以影响文本显示的位置, 使上下两行文字右对齐.

图 10 设置 unicode-bidi="isolate"之后的两行双向文本

4 文字位置的控制

上文提到, 使用 direction="rtl"可能改变文字显示的位置. 在实际使用中, 这是使用 direction 设置必须考虑的问题. 在未使用特殊设置的情况下, 文字都是从起始坐标点向右延伸的, 设置 direction="rtl"之后文字将从当前设定的位置向左边延伸, 与原来的情况恰好相反, 这会影响已有的 SVG 文本布局. 如果要确保 RTL 方式文字占据的空间位置跟 LTR 一样, 可以修改 text 元素的 x 值和 y 值, 一个更简单的方法是修改 text-anchor 的值, 使文本的起始点的相对位置发生变化, 代码形如

```
<text x="600" y="100" direction="ltr">…SVG 1.2…</text>
<text x="600" y="200" direction="rtl">…SVG 1.2…</text>
<text x="600" y="300" direction="rtl"
text-anchor="end">…SVG 1.2…</text>
```

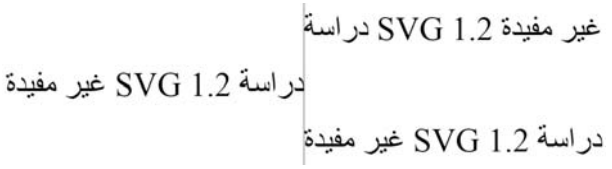

图 11 使用 text-anchor 对文本位置的控制

图 11 中的竖线标示了 x="600". 可以看到在从左到右的情况下, 文本以 x="600"为起始位置向右延伸, 在从右到左的情况下, 文本以 x="600"为起始位置向左延伸. 如果在从右到左的情况下设置了 text-anchor="end", 文本将以 x="600"为结束位置, 这样整个文本仍然可以显示在 x="600"竖线的右边, 所占据的空间位置跟从左到右的情况下一样, 这样在开发某些应用程序时比较方便.

如果希望文本不管被设置为从左到右还是从右到左, 占据的位置都相同, 那么可以将 text-anchor 设置为"middle", 代码形如

```
<text x="600" y="100" direction="ltr" text-anchor=
"middle">…SVG 1.2…</text>
<text x="600" y="200" direction="rtl" text-anchor=
"middle">…SVG 1.2…</text>
```

从图 12 可以看出, text-anchor="middle"的设置可以使文本占据的位置不随 direction 的改变发生变化.

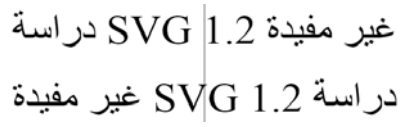

图 12 设置 text-anchor="middle"之后的显示位置

需要注意的是, 本文的讨论大部分基于 W3C 和 Unicode Consortium 协会的标准, Chrome 浏览器较好地支持这些标准, 然而不同的浏览器对标准的支持程度是有差异的, 对于不同的浏览器需要使用不同的处理方式. 在 Internet Explorer 11(简称 IE11)浏览器中, 双向文本在设置 text-anchor 值为"middle"或"end"的时候将可能出现文字重叠问题, 例如

```
<text x="600" y="100" direction="rtl"
text-anchor="start">…SVG 1.2…</text>
<text x="600" y="200" direction="rtl"
text-anchor="middle">…SVG 1.2…</text>
<text x="600" y="300" direction="rtl"
text-anchor="end">…SVG 1.2…</text>
```

在 IE 11 中将显示为图 13 的样子. 可以看到除了设置 text-anchor="start"之外, 其他的设置都出现了文字重叠问题. 这表明在 IE 11 浏览器中不能用 text-anchor 的设置来解决文本位置的控制问题, 只能使用其他方式, 例如改变文本元素的坐标值来控制文本位置. 通过与图 3 的对比, 还可以看出对同样的文本设置 direction="rtl", IE11 中的显示与 Chrome 浏览器中的显示不同, 这反映了不同浏览器中不同的 Unicode 双向算法实现.

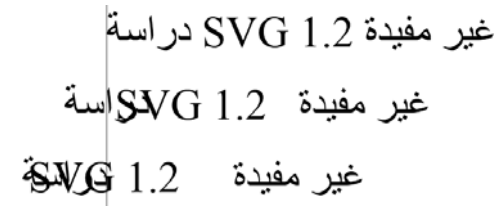

图 13 在 IE 11 浏览器中设置不同 text-anchor 值的双向文本显示

5 结论

在国际化软件中经常出现的双向文本的正确显示是一个比较复杂的问题, 解决方案的本质在于确定每

个字符的显示位置. SVG 在浏览器的实现中已经内置了 Unicode 双向算法的支持, 然而由于双向文本的特殊性, 仍然可能出现无法正确显示的情况, 需要开发者使用特定的设置告诉双向算法进行特殊处理. 为了在 SVG 中正确显示双向文本元素, 首先需要根据文本的内容确定是否需要设置 direction 属性值, 然后检查文本中的特殊字符, 特别是镜像字符是否被正确判断了方向, 可以对某些特殊字符使用 tspan 配合 unicode-bidi 属性的设定来影响 Unicode 算法对其方向的判断. 对于文本元素中使用了 tspan 改变某些文字样式和位置的情况, 可以通过对 tspan 的 unicode-bidi 属性设置来避免双向算法跨越 tspan 对文本进行重排. 此外还需要注意到设置了 direction 属性之后, 为了保持文本所占据的位置不变, text-anchor 的值可能需要进行正确的设定. 由于不同的浏览器对于标准的支持各不相同, 在不同的浏览器中, 可能需要使用不同的设定.

参考文献

1 Ishida R. Creating SVG tiny pages in Arabic, Hebrew and other right-to-left scripts 2011, https://www.w3.org/International/tutorials/svg-tiny-bidi/.

2 李培峰,朱巧明,钱培德.一个面向信息处理的双向文字处理算法 IBidi.计算机应用,2007,6:69.

3 Williams JL. Learning html5 Game Programming: A Hands-on Guide to Building Online Games Using Canvas, SVG, and WebGL. Addison-Wesley Professional, 2012.

4 刘旭.Chrome V8 引擎中的 JavaScript 数组实现分析与性能优化.计算机与现代化,2014,(10):66–70.

5 周琳,孔雷,赵方庆.生物大数据可视化的现状及挑战.科学通报(中文版),2015,60(5/6):547–557.

6 杨阳.微博内容的采集、分析及其可视化研究[硕士学位论文].大连:大连理工大学,2015.

7 杨立法.基于 SVG 的 Google 用户地图文本标注方法.昆明理工大学学报(自然科学版),2014,39(5):21–25.

8 Ishida R, Lanin A. Inline markup and bidirectional text in HTML. https://www.w3.org/International/articles/inline-bidi-markup/. 2014.

9 Consortium U. Unicode bidirectional algorithm. http://unicode.org/reports/tr9/. 2015.

10 Bettels J, Bishop FA. Unicode: A universal character code. Digital Technical Journal, 1993, 5(3): 21–31.

11 Davis M. Unicode bidirectional algorithm. Unicode Standard Annex, 2008, 9

12 Ishida R. Unicode bidirectional algorithm basics 2013, https://www.w3.org/International/articles/inline-bidi-markup/uba-basics.

13 买买提依明·哈斯木,吾守尔·斯拉木,维尼拉·木沙江,等. 基于统计专用字符的维、哈、柯文文种识别研究.中文信息学报,2015,29(2):111–117.

14 陈鸽,王廷梅,赵玮.一种新的维汉英混排文本显示模型的设计.硅谷,2012,(16):53–53.

15 Dürst M, Freytag A: Unicode in XML and other markup languages. Unicode Technical Report. 2002.

16 张梅静.基于 Android 平台的双向文本编辑及显示[硕士学位论文].成都:西南交通大学,2013.

17 Bellamy-Royds A, Cagle K. SVG Text Layout: Words as Art . O'Reilly Media, Inc., 2015.

18 Network MD. unicode-bidi-CSS|MDN. https://developper.mozilla.org/en-US/docs/Web/CSS/unicode-bidi. 2016.