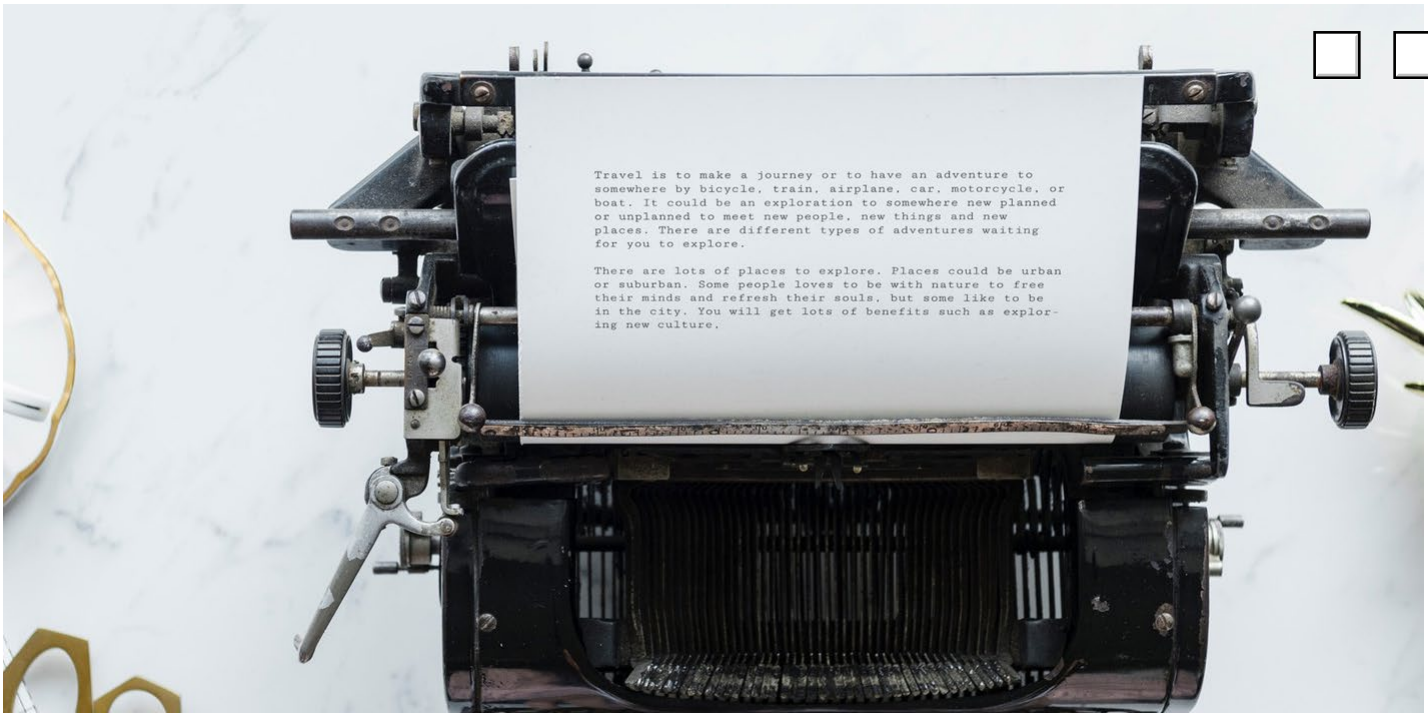


理解数字世界中的纸张：PDF | 科普

2018 年 09 月 12 日

PlatyHsu

引言

PDF 是我们打交道最多的文件格式之一。提到这个格式，即使是对技术并不熟悉的用户，也能说出「通用性好」、「格式不会乱变」这些优点。但同时，PDF 也是让我们感到困惑最多的格式之一，因为与 Word 文档等其他常见办公软件格式相比，PDF 似乎有着太多的「怪癖」，例如复制文字困难、几乎没法编辑等等。PDF 软件数量繁多、质量良莠不齐的现状，也进一步让很多用户无法正确理解和使用 PDF。

然而，事实并非如此。这些问题大多不是 PDF 格式的「缺陷」，而是因为我们在观念上把 PDF 当成了和其他办公文档格式相近的东西，并因此期待 PDF 也具有和后者相似的功能和特征。

尽管 PDF 格式和 Word 格式在实际用途上有诸多重叠之处，但那只是表面现象。从技术角度看，两种格式之间的差异要远远大于 Word 文档和网页之间的差异，甚至还要大于 Word 文档和 Excel 表格之间的差异。

但这并不意味着 PDF 就是一种难以理解的格式。恰恰相反，对大多数用户来说，PDF 可能是他们接触到的格式中最「接地气」、与现实生活最接近的。因为，PDF 与其说是一种数字文档，不如说是实体文档在数字世界中的影像。对 PDF 的操作，很大程度上可以看成对真实纸张的操作，只是操作环境从物理世界换到了数字世界而已。PDF 的创建就是一种虚拟的打印，复制 PDF 文字的过程更像是一种抄写，而 PDF 的编辑实质上是一种涂改。

一旦接受了这个观念，PDF 的很多「怪癖」就显得顺理成章了：打印出来的东西当然不会因为位置的变化而改变外观；抄写的结果很可能与原文存在误差，并且受制于抄写者对文字的理解；涂改的可行性则取决于原有布局留下了多少改动空间，并且再轻微的涂改也会对纸张造成损伤。

当然，只给出这样的结论并不足以令人信服。因此，下文的主要任务就是通过回答几个关于 PDF 格式普遍关心的问题，结合 PDF 的结构和语法，解释为什么「PDF 的本质就是数字化的纸张」，从而深化对 PDF 格式的理解。

为什么 PDF 的外观非常稳定?

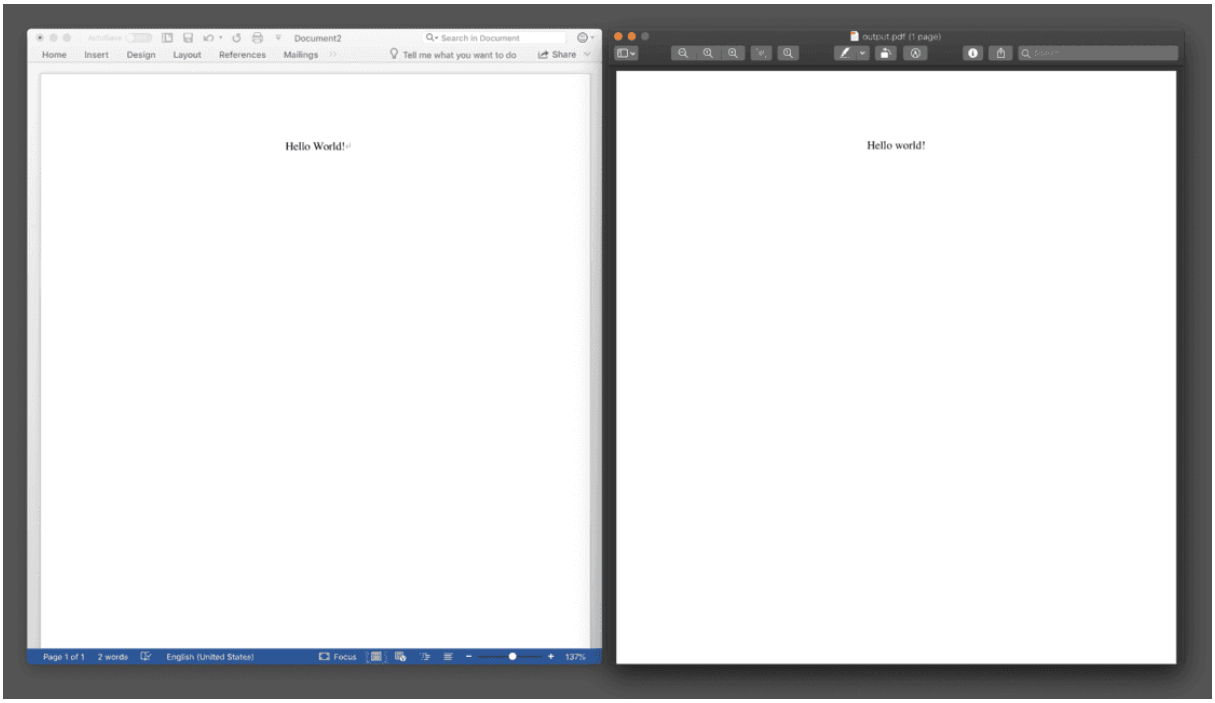
在我们日常使用的文档格式中，PDF 文件的外观是最稳定的。一经生成，无论在什么操作系统上、用什么软件打开，得到的显示效果几乎总是一致的。相比之下，Word 所使用 docx 格式的保真性就差得多了：哪怕只是换台电脑，显示效果都可能发生了变化，更不要说用不同版本的 Word 或者第三方软件打开了。

PDF 的这种保真性让它广受青睐，但它究竟是如何做到这一点的？PDF 的稳定是绝对的吗?

「打印」出来的 PDF

如果平时注意观察，容易发现各种软件中涉及 PDF 的操作，用词都比较特别。其他格式都是被「新建」（new / create）出来或者「保存」（save）下来的，只有 PDF 是被「导出」（export）甚至「打印」（print）出来的。这些词语并不是随意选用的，它们本身就说明了 PDF 的重要特征：「导出」暗示着文件编辑已经告一段落，而「打印」则更是形象地表明 PDF 的创建是一个「固化」的过程。

为了进一步理解这种区别，让我们来对比一组外观上完全相同的 Word 文档和 PDF 文档。下图中，右侧的 PDF 格式文件是由左侧的 Word 文件导出得到的，两者的内容都只有「Hello world!」一行居中文字。（注：为了避免不必要的复杂性，我们暂时只以英文文档为例。）



一对内容相同的 Word 文档和 PDF 文档

先来看 Word 是如何实现这一版式效果的。用任意解压工具将该文档解开（docx 文件实质上 就是一个压缩包），找到其中的 `/word/_rels/document.xml` 并打开。其中的关键部分如下（代码经过整理）：

```
<w:pPr>
    <w:jc w:val="center"/>
    <w:rPr><w:rFonts w:ascii="Times" w:hAnsi="Times"/><w:lang w:val="en-US"/></w:rPr>
</w:pPr>
<w:r w:rsidRPr="003C75CF">
    <w:rPr><w:rFonts w:ascii="Times" w:hAnsi="Times"/></w:rPr>
    <w:t>Hello world!</w:t>
</w:r>
```

不必恐惧这些陌生的代码，它们的意义很容易从文档内容本身反推出来。先看倒数第二行，这里记录了整个文档最重要的信息——「Hello world!」这串文字。在它的上面一行，`w:rFonts` 属性将字体设置为 Times。那么前四行是做什么的呢？从第二行的 `center` 字样不难猜出，它们控制的是文字所在段落的样式，包括居中对齐等等。

这就是 docx 格式这类 标记语言 （Markup Language）文档的特征：在纯文本上包裹各种「标签」（tag）来描述文本的样式（颜色、位置、字体等等），从而获得格式丰富多变的文档。常见的网页（HTML）、Evernote 笔记（ ENML ）所用的语法本质上都是标记语言，区别只在于支持的标签各不相同、因此能实现的格式有多有少罢了。

PDF 又是怎么做的呢？我们用纯文本编辑器打开上图中的 PDF 文件（是的，PDF 可以用文本编辑器打开查看源码），其中的关键部分如下（经过处理）：

```
BT
    1 0 0 1 1036 572 Tm
    /TT1 12 Tf
    [ (He) 24 (l) -48 (l) -48 (o) ] TJ
ET
BT
    1 0 0 1 1147 572 Tm
    /TT1 12 Tf
    ( ) Tj
ET
BT
    1 0 0 1 1160 572 Tm
    /TT1 12 Tf
    [ (w) 24 (or) -84 (l) -24 (d) ] TJ
ET
```

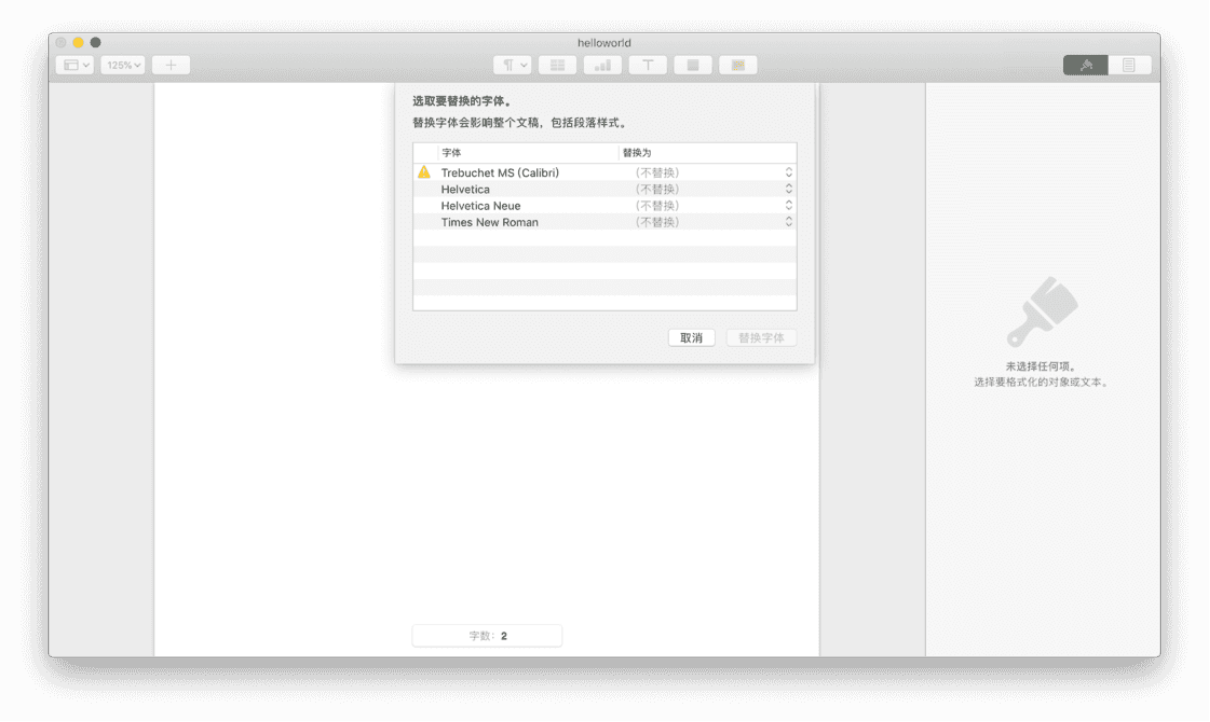
即使你被这些凌乱的数字和代号弄得一头雾水，大概也能看出它与 docx 格式有着截然不同的画风。如果我们粗略地把这些语句翻译成「人话」：

```
【文字开始】
```

```
    缩放比例1倍 坐标(1036,572) 【文字定位】
    /TT1 12磅 【选择字体】
    [ (He) 间距24 (l) 间距-48 (l) 间距-48 (o) ] 【绘制文字】
【文字结束】
【文字开始】
    缩放比例1倍 坐标(1147,572) 【文字定位】
    /TT1 12磅 【选择字体】
    (空格) 【绘制文字】
【文字结束】
【文字开始】
    缩放比例1倍 坐标(1060,572) 【文字定位】
    /TT1 12磅 【选择字体】
    [ (w) 间距24 (or) 间距-84 (l) 间距-24 (d) ] 【绘制文字】
【文字结束】
```

请在头脑中想象一下这个过程——是不是有一种强烈的「机械感」？如果说 Word 文档使用的标记语言很像是给人下命令（「这里有一段话，把它用 Times 字体写出来，位置上居中对齐……」），那么 PDF 的语言则更像是在控制机器，定位、调整、落笔、抬起，移动到下一行；如此重复。联想一下，铅字排版和打印机的工作机制不也是类似的吗？可见，用「打印」一词来搭配 PDF 是十分恰当的。

如此对比之下，PDF 显示效果的保真性就容易解释了。虽然 Word 格式使用的语法明显更容易理解，但问题在于不同「人」——不同软件环境——听到同样一段指令，头脑中的反应未必是相同的。「用 Times 字体显示」——哪个是 Times 字体？没有安装这个字体怎么办？「居中对齐」——以什么为参照物居中？怎么计算居中？Word 文档对此笑而不语，把问题留给了软件去思考。正是这种自由裁量的空间为显示效果的差异留下了隐患。



用 Pages 打开 Word 文档出现的字体缺失问题

在 PDF 中，这样的问题就很难出现。与标记语言那种相对的、描述式的标签不同，PDF 语言几乎都是绝对的、指令式的。例如，上面的 PDF 中，文字明显是居中的，但代码中从头到尾没有半个字提到「居中」；相反，它直接指明了文字的坐标。因此，无论什么阅读器读到这份文件，只要根据坐标「照葫芦画瓢」地绘制，一定能得到相同的显示效果。

类似地，下面这段代码的作用是绘制出一个点状的「L」形，很难想象它会给软件留下什么「自由发挥」的空间：

```
8 w 1 J [20] 0 d【设置画笔为 8pt 宽、末端圆角、20pt 点状线】
200 200 m 200 100 l 250 100 l 【移动到(200,200) 画直线到(200,100) 画直线到(250,100) 】
S 【确认绘制】
```

PDF 外观稳定性的另一个原因是它嵌入了各种需要用到的外部资源。我们注意到，上面的 PDF 在设置字体时，没有像 docx 文件那样直接指定字体名称，而是引用了一个代号般的 `/TT1`。实际上，`/TT1` 指向的是一个内嵌的字体，存储在 PDF 内部的偏后位置。显示时，阅读器将根据这个代号找到字体，按照其中记载的形状、宽度等信息，将字符「书写」在指定的坐标上——就好比活字排版工按指示从字盘中取出字模，并放在母版上的特定位置一样。相反，Word 文档默认是不嵌入字体的；只要另一台电

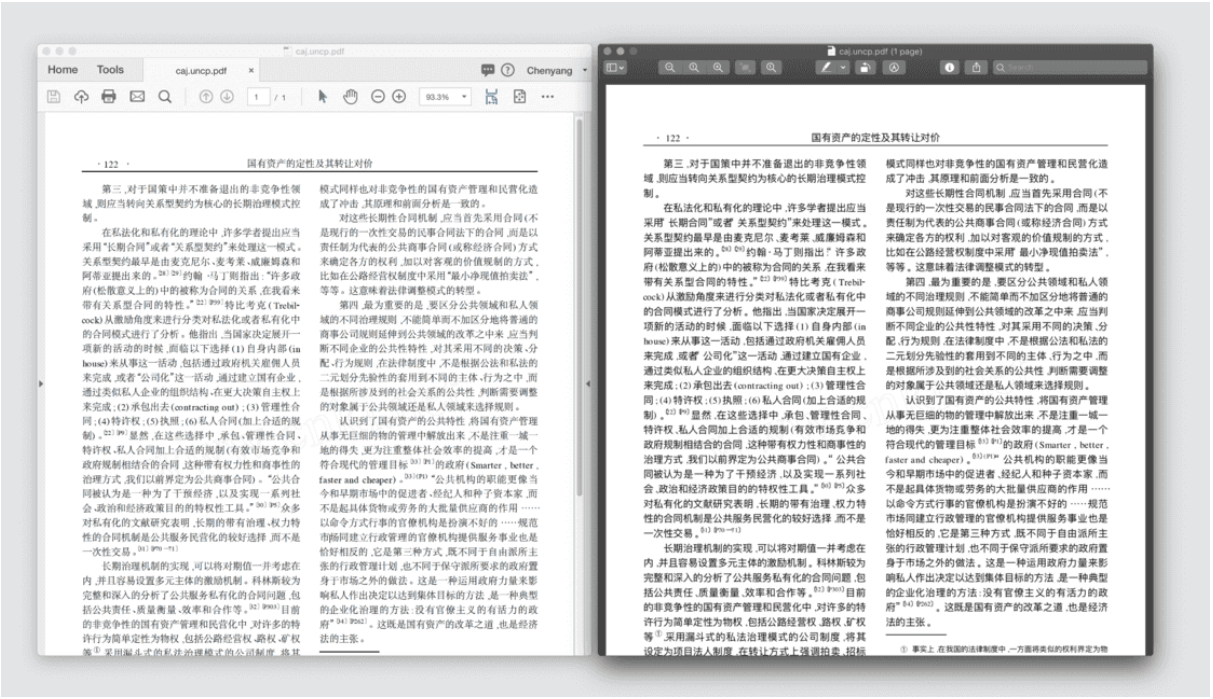
脑上没有安装用到的字体，或者安装了不同版本的字体，就会导致显示效果的差异。

活字

保真性的例外

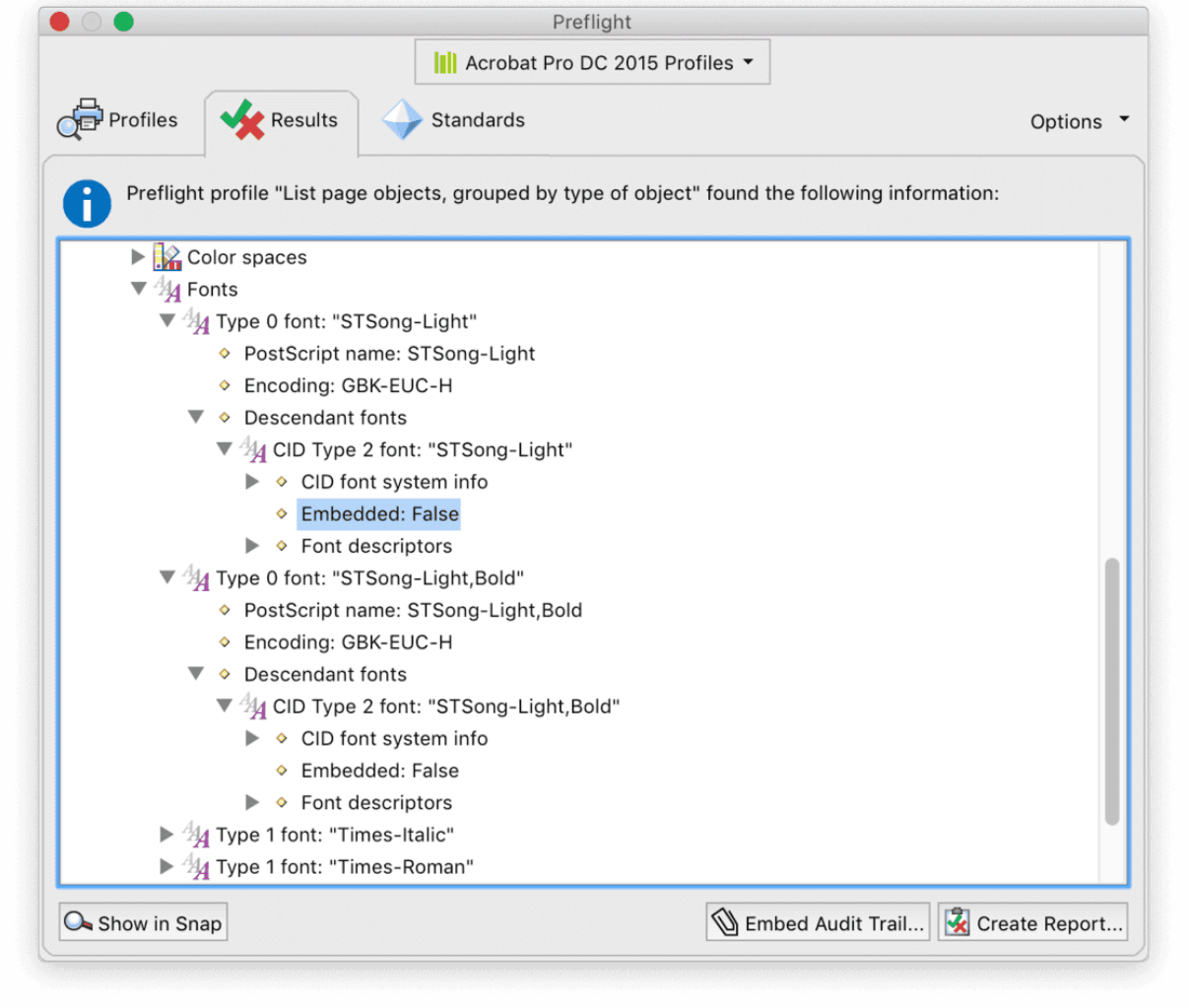
不过，PDF 的这种保真性也不是绝对的。尽管 PDF 语句已经非常精确，也不能排除存在不同解读方式的可能。特别是在 PDF 文件本身有瑕疵（例如语法不合规范、资源文件缺损等）时，阅读器就必须「猜测」文件原本的意图，误差也就因此产生了。

例如，下面这份 PDF 格式论文是从中国知网下载的。在用 Adobe 的官方 PDF 工具 Acrobat 打开时，文件基本显示正常。而在用 macOS 自带的预览 app 或 PDF Expert 打开时，就能明显看出显示效果是有问题的——所有的中文都变成了黑体，而不是应有的宋体。（你可能注意到从知网下载的 PDF 经常存在字体显示问题。这是因为它们并非由排版软件直接导出的「一手」版本，而是从私有的 CAJ 格式转制而来的，存在很多兼容性问题。）



存在字体显示问题的知网 PDF

为什么会出现这样的问题呢？用 Acrobat 附带的 Preflight 工具检查文件，可以发现它引用了包含 `STSong-Light` （即华文宋体细版）在内的数个字体，但却没有嵌入对应的字体文件。显然，这种空头支票式的做法会引起阅读器的困惑，它们将只能「猜测」该用什么来代替 `STSong-Light`，不同的猜测结论将引发不同的显示结果。



检查发现问题 PDF 没有嵌入应有的中文字体

其中，Acrobat 是一个非常「仁慈」的阅读器，会尽可能做一些合理推测来修复文件本身的瑕疵。可以看出，它选择了一个类似的字体——同属宋体的 Adobe Song Std Light——来代替这个不存在的 `STSong-Light`。相反，自带的预览 app 就没有那么多考虑了，它选择直接回退到系统界面的默认字体—— 苹方 来显示，于是我们就只能和一片黑糊糊的黑体大眼瞪小眼。而如果用 Acrobat 的修复功能将缺失的字体文件补充进 PDF 后，将会发现预览 app 也能正确显示中文字体了，这印证了我们对问题成因的分析。

为什么 PDF 中的文字经常难以复制?

如果说 PDF 显示效果的稳定是它吸引人们使用的主要优势，那么「文字难复制」一定是它让很多人敬而远之的原因之首。确实，从网页上复制文字的那种方便灵活在 PDF 这里几乎是一种奢望。很多时候，就连第一关——准确选中要复制的文字，都是难以

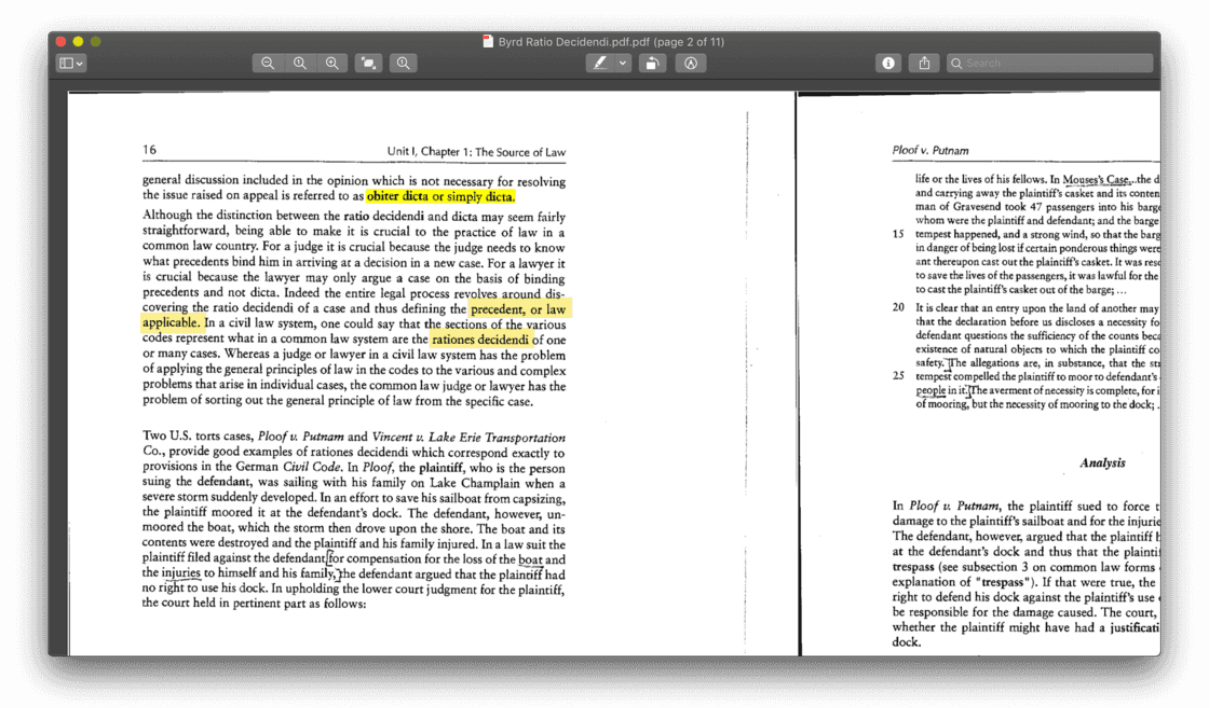
跨越的障碍。

PDF 为什么对复制操作这么不友好呢？换种问法，满足什么条件的 PDF 才能正确复制出文字呢？答案是，PDF 文件的内容、文本的排列方式，以及文本和字体的编码都会影响到文本的复制。另外，阅读器对于复制操作的优化也是不可忽视的。

从扫描版 PDF 复制文本

要把文字复制出来，一个前提是 PDF 中必须真的包含文字。这听起来好像是一句废话，但实践中大多数「文字复制不出来」的问题，原因都是目标 PDF 中根本没有文本。这尤其多见于那些扫描而来的电子书和电子文档；如果不经处理，它们的每一页不过是原始文件的一张「照片」而已，显然无法复制出文字。

当然，扫描版 PDF 是完全可能支持复制的。OCR（光学字符识别）工具可以识别出图像中的文本，并将其写入 PDF 中，使其支持选中和复制；这也是很多收费 PDF 软件主要宣传的功能点。需要避免的一个认识误区是，即使经过 OCR 处理的 PDF，其中的文本也并不是「附身」在原来的图片上的。相反，它们通常被存储在一个单独的文本对象中，与图片中的文字位置一一对齐。只不过由于 PDF 中的文本可以被设置为隐藏，当用户试图选中和复制这些隐藏文本时，看起来就好像是直接在图片上选中和复制文字一样。

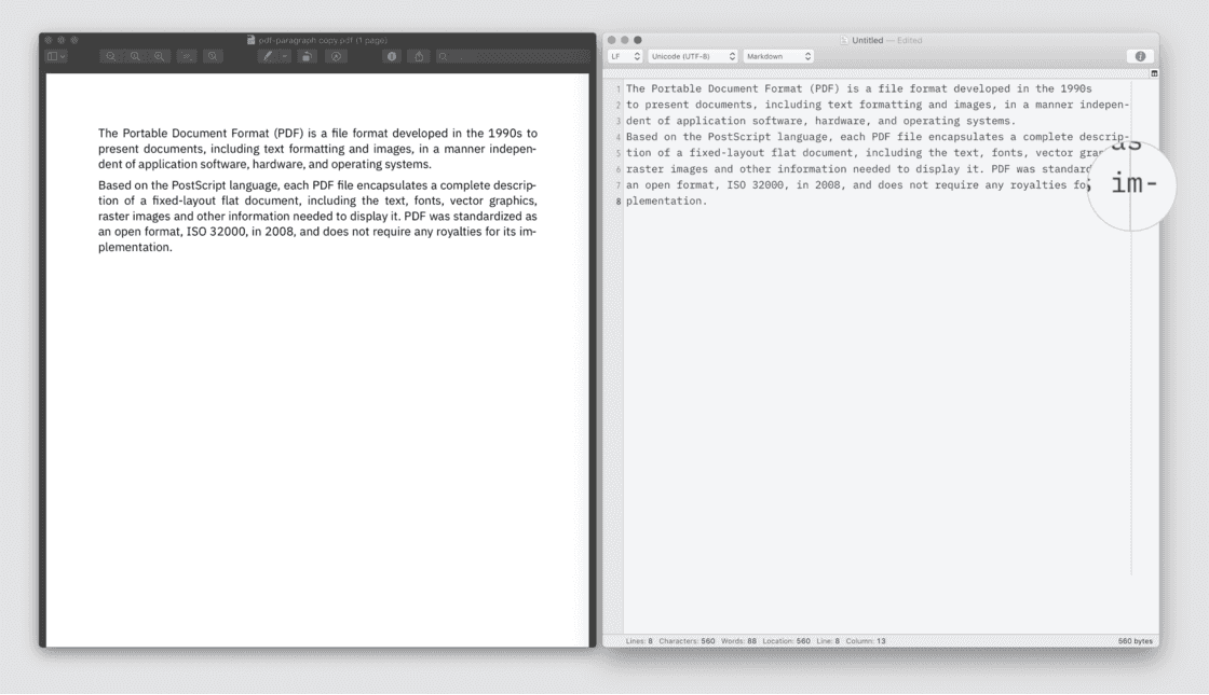


经过 OCR 处理的扫描版 PDF 同样可以选中和复制

这就解释了为什么在一些扫描版 PDF 中，文字选中效果看起来是「歪」的；而在另一些 PDF 中，选中和复制操作总是不连续或不完整。前者是因为加入的隐藏文本图层没有与图片上的文字位置对齐，后者则是因为 OCR 识别不全，或者将连续的文本识别成了孤立的文字。

跨行和跨段复制文本

麻烦还远没有结束。即使确认了 PDF 中包含文本，也并不意味着就能把它们原封不动地复制出来。在复制跨越数行乃至数段的大片文本时，我们往往不能得到所预期的完整段落，而是多了一些无用的空白或者换行。例如，在下图所示的复制结果中，PDF 似乎把页面上的换行和连字符照单全收了，即使它们显然不是原文的一部分。同样的问题也会影响到文本搜索，很多 PDF 中，断成两行的词是无法正常搜索的。



复制操作容易把不需要的内容也一并复制出来

这类现象仍然可以从「纸」的比喻中得到解释。和纸质文件一样，PDF 只负责记载形态而不记载含义。一个可能违反直觉的事实是，PDF 对于「段落」是没有概念的。在其他格式中复制跨越两行的文本，之所以能得到连续的结果，是因为软件知道这串文字属于同一个段落，显示成两行只是因为文档宽度的限制。类似地，如果一个长单词被拆成两行显示，其他格式在复制时是不会带上连字符的，因为软件知道那不是单词的一部分。

在 PDF 中，这些结论都不适用。还是以上面那份 PDF 为例。观察它的第一段第二行到第二段第一行：这三行涉及了分词（行尾连字符）和分段两个排版要素。这一部分在 PDF 代码中的反映是：

```
q 1 0 0 1 57 755 cm BT 【定位到(57,755)】
    13 0 0 13 0 0 Tm /Tc1 1 Tf 【设定字体】
    [(…略去之前部分…) -141.7 (indepen) ] TJ 【绘制第一段第二行】
ET Q
q 1 0 0 1 533 755 cm BT 【定位到(533,755)】
    13 0 0 13 0 0 Tm /Tc1 1 Tf 【设定字体】
    (-) Tj 【绘制行尾的连字符】
ET Q
q 1 0 0 1 57 738 cm BT 【定位到(57,738)】
    13 0 0 13 0 0 Tm /Tc1 1 Tf 【设定字体】
    [ (dent o) 6 (…略去中间部分…) (ems) 5 (. ) ] TJ 【绘制第一段第三行】
ET Q
q 1 0 0 1 57 715 cm BT 【定位到(57,715)】
    13 0 0 13 0 0 Tm /Tc1 1 Tf 【设定字体】
    [ (Based ) (…略去之后部分…)] TJ 【绘制第二段第一行】
ET Q
```

这里，我们需要关注的只有两点：第一，代码中的 `Tj` 或 `TJ`（绘制文字）命令每次处理的文字都不超过一行。实际上，PDF 实现换行和分段的方法，只不过是把那个「想象中的笔尖」移动到下一行或下一段的起始坐标处罢了。换句话说，除了笔尖移动幅度更大（以反映段间距），分段和普通分行在 PDF 中没有任何区别。

第二，原文第二行行尾的连字符出现在了 PDF 代码中。在同一份文档的 Word 版本代码中，这个连字符是不存在的，因为它并不是输入内容的一部分。换句话说，其他格式不在文件中记录，只在打开时才根据需要显示的排版特性，制成 PDF 后却被「固化」下来了。

既然 PDF 不记录段落信息，而换行和连字符又明明白白地写在文件中，它们会出现在复制结果里也就并不奇怪了。

当然，阅读器可以根据文本的内容和布局等特征进行推断，提供更为合理的复制效果。例如打开上面的 PDF 时，PDF Expert 和 Chrome 浏览器内建的阅读器都会对行尾的连字符作特殊处理，复制的结果是一个完整的单词，搜索中间断开的「independent」整词也能定位到结果；相反，用 macOS 自带的预览 app 和 Firefox 内建的阅读器打开时，复制操作会把连字符当作单词的一部分，搜索整词也得不到结果。

可能有人会问：PDF 为什么连如此基础的段落功能都不支持呢？这实际上是一种有意为之的选择。PDF 是作为一种电子化的打印被 发明 出来的，而打印意味着编辑工作的终点。因此，PDF 完全可以不考虑宏观层面的排版功能（因为用来创建它的字处理和制版软件在这方面更专业），而只保留对少数局部版式（例如文字间距/Kerning）的支持。这不仅有利于节省文件空间，也进一步强化了 PDF 跨平台显示的稳定性。

复制 PDF 文本的乱码问题

确认了 PDF 里有文字，文本的排列也很整齐，是不是就能皆大欢喜了呢？很遗憾，对于中文用户来说，从 PDF 中复制文字经常还会遇到一道坎：乱码。

乱码问题虽然在其他场合（例如网页、游戏等）也存在，但那基本都是发生在文字的显示阶段。既然 PDF 都正确显示了文字，为什么一经复制却变得不是那回事了呢？

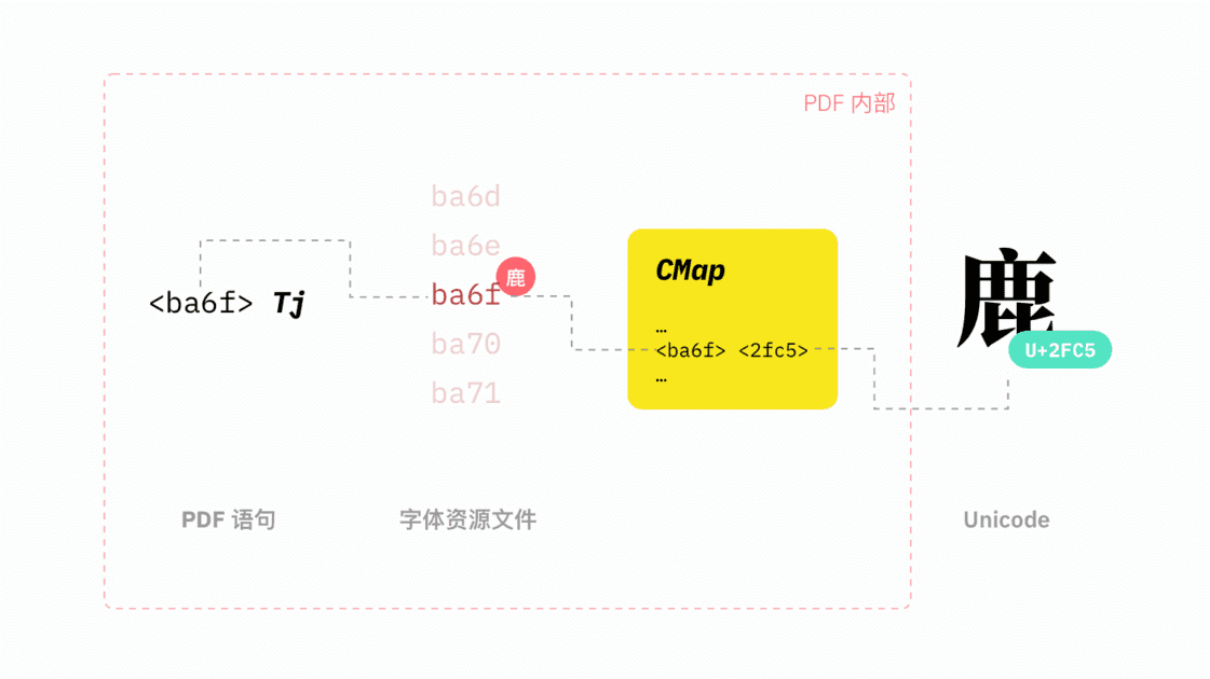
如上所述，PDF 的生成本质上就是一个「虚拟打印」的过程。这个比喻的言外之意——打印机是不识字的。它的全部使命，就是在纸张上按照坐标和字体记载的形状，照葫芦画瓢地排列出指定的文字；整个过程完全不涉及「这是什么字」之类的问题。类似地，PDF 绘制文本的流程，也不过是从字体资源文件中取出特定码位上的字符（glyph），放置在某个坐标位置上。至于取出来的到底是什么字，PDF 根本不关心。

可如果 PDF 是一个「文盲」，它在复制操作中又是靠什么给出我们需要的文本的呢？

我们知道，计算机中的文字是通过编码来区分和调用的。目前，最通用和常见的编码方案是 Unicode。另一方面，在 PDF 内部嵌入的字体文件中，每个字符也有一个编码；绘制文字时，`Tj`/`TJ` 命令可以通过这个编码向字体文件索取需要的字符。

上述两套编码可以是（并且在非英文环境下往往是）不同的，它们之间的「桥梁」，就是字体所附带的 `ToUnicode` 属性。`ToUnicode` 的值是一张映射关系表（称作

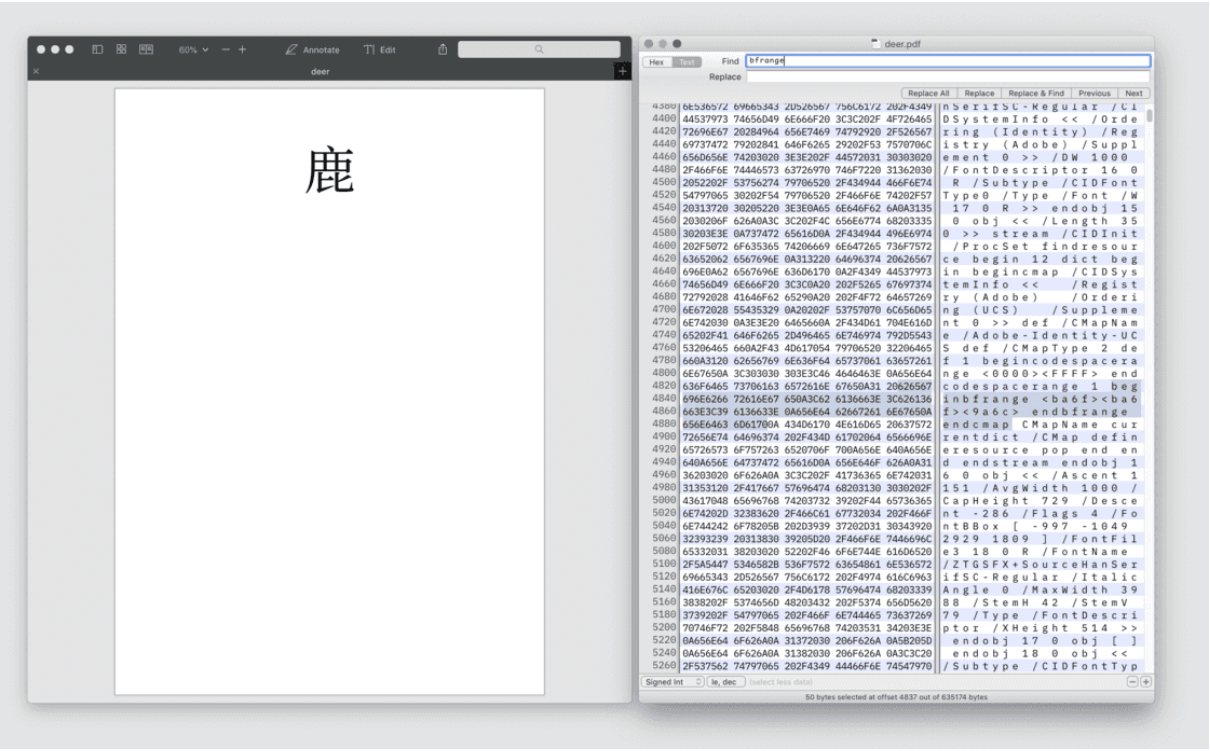
character map / CMap），它给出了字体文件中每个码位上的字符与 Unicode 编码的对应关系。在复制、搜索等操作中，PDF 就是靠 CMap 这张表得知每个字符到底是什么「字」的。



PDF 涉及的两套编码

问题是：如果这张表弄丢了呢？或者虽然没有丢，但是缺损了一块？或者被人恶意篡改了？

让我们来做一个「指鹿为马」的实验。下图是一个非常简单的 PDF 文件，上面只有一个「鹿」字。



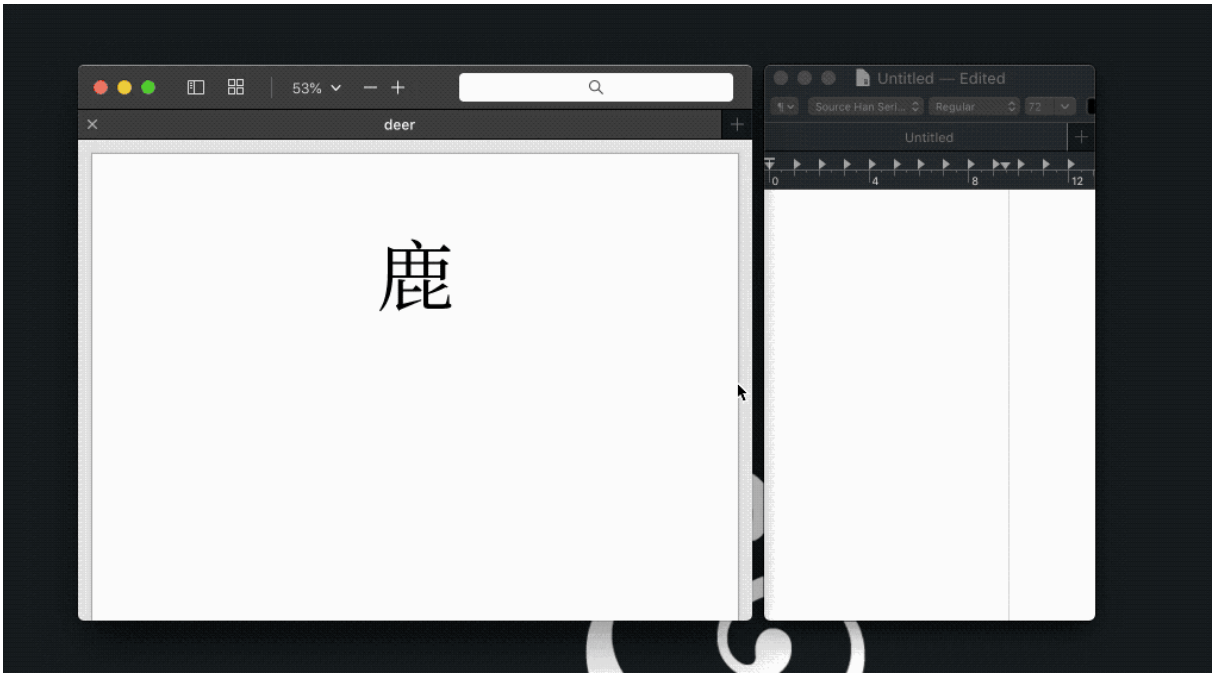
示例 PDF 和它的 CMap 片段

用文本编辑器打开 PDF，可以看到 `<ba6f> Tj` 这一语句。不难猜出，该语句的作用是绘制字体文件中编码为 `0xBA6F` 的字符。根据 PDF 对象的交叉引用关系（后文会介绍）找到 CMap，其关键部分为：

```
1 beginbfrange
<ba6f> <ba6f> <2fc5>
endbfrange
```

不难看出，这个 CMap 告诉阅读器：字体文件中编码为 `0x6ABF` 的字符对应 Unicode 中的 `U+2FC5`。查 Unicode 码表可知，`U+2FC5` 正是汉字「鹿」。

现在，我们将上面代码中的 `2FC5` 改成 `9A6C`，然后重新打开 PDF 文件。外观上，文件似乎并没有任何变化。但如果你试着把这个字复制出来，会发现得到的结果是「马」。



「指鹿为马」

这并不是什么灵异事件，因为「马」在 Unicode 中正是 `U+9A6C`。

上述实验表明，PDF 中的「所见」未必就是「所得」。只要 `/ToUnicode` 的内容（较旧的或者英文的 PDF 也可能是 `/Encoding` 属性）发生丢失或者错误，文字复制就会出现乱码。与此同时，PDF 的显示却不会受到任何影响（因为文字绘制是由与编码完全无关的语句控制的）。反过来思考，利用乱码故障的原理，也可以通过故意破坏或者修改 CMap，制造出表面正常但完全无法复制和搜索的文件，间接达到加密的效果。

为什么 PDF 很难编辑?

除了显示、复制上的问题，PDF 难以编辑的特性也经常引发用户的疑问。不仅大多数可以免费获得的 PDF 工具都只有阅读功能，即使是那些具有编辑功能的软件，编辑效果也往往达不到预期，甚至反而影响其他部分的格式。

如何解释 PDF 的这一特征呢？还是回到本文一以贯之的思路：把 PDF 看作真实的纸张来理解。

试想你手上有一份纸质笔记或者文件需要临时涂改。生活经验告诉我们下面几点事实：第一，涂改操作只可能是小范围的、逐字逐词的。即使真的不得不修改一大段内容，你也只能慢慢用涂改液或者胶带把它们抹去。第二，纸上原有的内容越多、格式越复杂，涂改起来就越困难。在增删文字时必须要考虑到前后文的限制；原来的内容

越密集，留下的修改余地就越小。第三，很难实现完全不留痕迹的涂改。任何涂改操作都会对纸张造成损伤；对文字的修正，从纸张的角度看反而是一种「污染」。

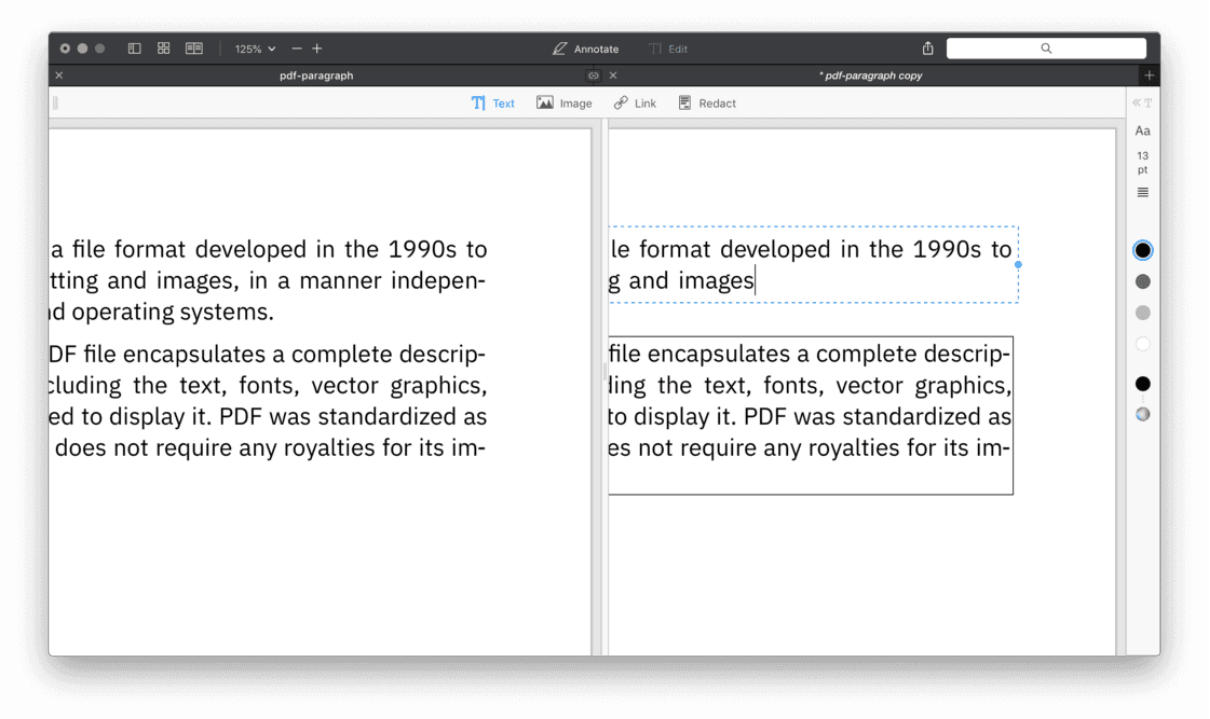
这些描述几乎完全适用于 PDF 文件的编辑，而原因还是要从内部结构来分析。前文中，我们已经了解了 PDF 语句如何在局部层面绘制出文本和图形，这里再从宏观角度简单介绍一下 PDF 的构造。

一个 PDF 文件主要由四个部分构成，依次是文件头（Header）、主体（Body）、交叉引用表（Cross-Reference Table）和文件尾（Trailer）。在其中最重要的主体部分，页面、文本等元素和字体、图片等资源存储在称为「对象」（object）的单元中。每个对象都有唯一的编号，并且可以无尽地相互包含和引用。例如，一个编号为 2 的页面对象可以包含一个编号为 3 的子对象来描述页面尺寸，又包含一个编号为 4 的子对象来存储该页上的文本；后者在指定字体时，又可以引用一个编号为 20 的字体对象，而这个字体对象的大小、字符宽度、曲线轮廓等信息也是分别存储在其他对象中的；等等。

一个可能的 PDF 的构造

读取 PDF 时，阅读器首先从文件头确定文件类型和版本号，旋即跳转到文件尾，获取交叉引用表的位置（以字节位置表示），它进而列出了 PDF 中所有对象的位置。凭借这张表，阅读器就能找到每个对象，解析它们之间的包含和引用关系，并按照其中的命令将文件的全貌绘制出来。

这样的构造对 PDF 的编辑有什么潜在影响呢？可以看出，PDF 的结构是高度固化并且相互依赖的。它就像是一个积木堆，其中的每一块「积木」——PDF 中的一个对象——都不是独立的，而是与四周的其他积木相互支撑。编辑 PDF 文件就如同试图改变一个成型的积木堆：移去或挪动一块积木（对象），周围的木块不仅不会自动补上空白，反而可能因为失去支撑（对象的交叉引用关系）而变形。比如，从 PDF 中删去一段，后面的文本并不会自动调整位置；相反，如果绘制它们的语句引用了被删除部分的样式，这些样式也可能随着删除操作而丢失。



用 PDF Expert 删除一段文字，后面的段落无法自动补齐空隙

而如果要往积木堆上增加一块——往 PDF 中增加内容，面临的风险同样很大。且不论现有的空隙是否允许这么做，你也无法预知剩余的积木块中是否有自己需要的。例如，出于节省体积的考虑，PDF 中嵌入的字体文件往往都是高度「子集化」的，只包含文件中用上的那部分字符。如果准备追加的字恰好不在其列，就很可能引发显示问题。

退一步说，即使编辑操作幸运地没有引发任何问题，它的成本也是很高的。哪怕只是插入一个字母的「微量」编辑，也会导致排在它之后所有内容的地址向后偏移 1 字节，于是依靠字节计数来定位的交叉引用表必须整个重写。假如你的改动幅度更大（例如用了新的字体），就需要靠新增对象来实现，于是其他对象也必须相应更新以反映对象编号的变化。另一方面，从 PDF 中删去内容时，编辑器未必能聪明到把不再有用的对象一并删去。很多时候，这些成为空壳的对象就被「抛弃」在原地，白白占用空间，并且增加阅读器解析文件时的计算成本。

作为对比，Word 格式那种标记语言的特性——在纯文本上包裹标签来记载格式信息——决定了它对于编辑操作是十分友好的。由于文本的内容和格式相互独立，修改文本内容并不会对格式造成影响，反之亦然。这就好比把水倒在有造型的容器中，换掉一部分水，剩下的水依然服从于容器的形状；换一个容器，水的形状随之变化，但还是原来那些水。

需要加以区分的是，对 PDF 的标注（annotation）操作——包括高亮、下划线、笔记

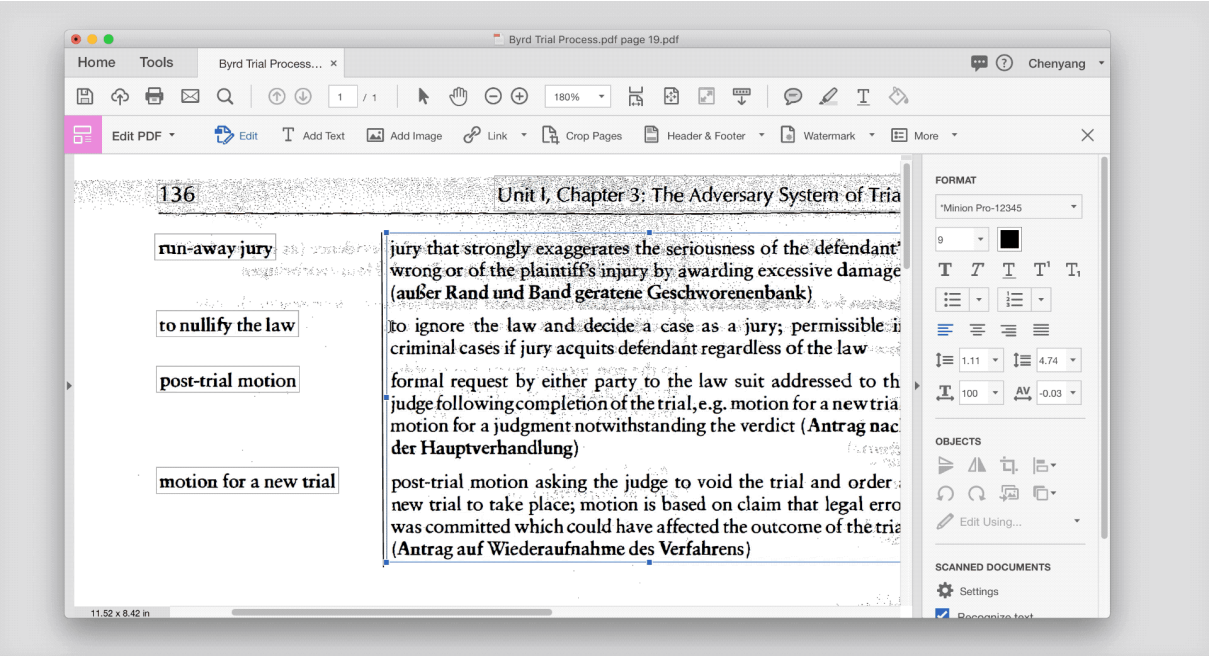
等——不属于「编辑」的范畴。在实现层面，PDF 中的标注是附属于所在页面的子对象，其中记载了标注的类型、位置、形状（如果有）、文本（如果有）等，与存储文件内容的对象相互独立。它们就像是纸上的便利贴，可以随时移除而不留下痕迹。

那作为用户，如果确实遇到编辑 PDF 的需求，应当如何解决呢?

首先，应当考虑是不是真的需要修改 PDF 文件本身。假如你将一份文件打印出来以后发现了错别字，第一反应恐怕是回到电脑上修改、然后重新打印那一页，而不是用胶带粘去错字然后手写。类似地，既然 PDF 文件的本质就是「电子纸张」，如果发现错误，最正确和简捷的做法应该是改动用于生成这个 PDF 的原始文档，然后重新导出一遍，而不是考虑怎么修改 PDF 本身。

即使手上没有生成 PDF 的原始文档，在直接修改 PDF 时也应该尽量控制编辑幅度。因为修改越多，对文件的「污染」就越大，也就越有可能造成格式混乱、体积膨胀等结果。如果要修改的内容确实很多，甚至可以考虑转换/ OCR 为其他格式—编辑文字—导出为 PDF 这样的路径，或许效果反而比直接编辑好得多。

最后，尽管 PDF 对编辑操作非常不友好，但毕竟「事在人为」，不同软件的编辑能力有很大差别。例如，来自第一方的 Acrobat Pro 就明显高于平均水平。根据测试，它不仅能从 PDF 的布局中判断出段落并以此为单位编辑（而不是孤立的文本块），还能在编辑中一定程度上维持原有的对齐方式、段间距等设置。更为「黑科技」的是 Acrobat 的 OCR 功能，它甚至可以做到在识别文字的同时，将文本矢量化后、分离到与背景独立的图层中，从而能增删扫描版 PDF 中的文字。



Acrobat 可以将扫面版 PDF 中的文字矢量化后实现编辑

结语

任何文件格式都有自己最擅长的用途。对 PDF 来说，它擅长的领域就是跨平台交换和文件归档，而那些需要频繁编辑文本内容和版式的应用场景，则是其不能胜任的。只是在日常使用中，我们常常被 PDF 和其他文档格式在外观上的相似所误导，把它用在了不擅长的领域，然后反过来抱怨这种格式在编辑和复制中的「笨拙」。这多少是错怪了 PDF。本文之所以一再将 PDF 类比为实体文档，一方面是为了便于解释技术原理，另一方面也是为了提供一种选用 PDF 格式的标准：适合打印出来的内容，一般也才适合「打印」成 PDF。

理解 PDF 的原理也有助于挑选合适的阅读/编辑工具。如上所述，软件对瑕疵 PDF 的宽容度和修复能力，对文本搜索、复制的识别、优化能力等细节，是最值得重点考察的；这些看似不起眼的功能点对使用体验和工作效率有极大影响。相反，那些频繁被当作营销「亮点」的功能，特别是 PDF 编辑、格式转换等，反而不那么重要。因为 PDF 从结构上就不适合修改，这些所谓的编辑功能很难达到用户的预期，何况还有大量类似于 SmallPDF 的免费工具可以满足临时的、精度不高的编辑需求。至于严肃、专业的 PDF 编辑，Acrobat 可能是唯一的选择。

最后需要说明的是，PDF 涉及的技术非常复杂，这篇文章只是从日常使用的角度做了最粗浅的介绍。文中很多地方为了便于理解，采用的解释和比喻是过度简化的。如果对 PDF 格式的原理有进一步的兴趣，建议直接阅读 PDF 的 标准文件。这份标准虽然十分冗长，但并不难读。哪怕只是挑选几个关心的主题来浏览，相信都会对理解 PDF 格式以至排版技术有很大的启发。

延伸阅读：

- PDF Reference and Adobe Extensions to the PDF Specification: Adobe 的 PDF 文档专页，提供了 PDF 1.7（成为 ISO 标准的版本）和 Adobe 基于 PDF 1.7 版所做功能扩展的文档。
- Understanding the PDF File Format: 帮助理解 PDF 格式的详尽系列文章，每篇都不长，语言比较浅显易懂。
- Planet PDF: Foxit 旗下的 PDF 专题网站。页面设计很老旧，但是能找到不

少探究 PDF 格式细节的资料。

- John Whitington, *PDF Explained* (O'Reilly Media, 2011): 动物园丛书里介绍 PDF 的不止一本，这本年代偏久远，但是个人认为是相比之下讲得比较清晰的。
- GitHub – PDF 101 – Learn and Play with PDF Source Code: 一个非常有意思的 GitHub 仓库，里面提供了大量手敲代码制成的 PDF，用文本编辑器打开按照里面的注释就可以像玩游戏一样做各种「实验」，对理解 PDF 格式非常有帮助。

> 下载少数派 客户端、关注 少数派公众号 ，让你的数字生活更精彩

热门文章 # 文件管理 # 文档编辑

822 等 822 人为本文章充电