PDF テキスト抽出についての調査結果

2008-03-02
治郎吉商店　山本 卓

1. はじめに

PDF ファイルからテキストを抽出するために必要な処理に関する情報を記します。まずは PDF というファイルフォーマットについて、テキスト抽出という観点から簡単に説明します。続いてこのフォーマットを解析するのに必要な処理を整理します。そして、実装を行う上でのポイントを整理します。

なお、本文書中に出てくる/.../といったスラッシュで囲った等幅フォントの表現は、テキスト書式を表す正規表現です。

2. PDF ファイルフォーマット

2.1. 概要

PDF は Portable Document File の略であり、印刷用ページレイアウト記述言語 Post Script を拡張し、圧縮を取り入れることでファイルサイズの低減を図った言語です。Post Script はプレーンテキスト形式ですが PDF は圧縮を取り入れたため、PDF は基本的にテキスト形式でありながら内容を圧縮した結果のバイナリが入っている、テキストファイルとバイナリファイルを合わせたフォーマットになっています。なお、基本がテキストファイルであるため改行コードは CR、LF、CRLF の三種類が存在することにも注意してください。

PDF は Post Script 同様にページレイアウトを記述する言語です。したがってリッチテキストなどが持つ「文書」に適した構造ではなく、Illustrator ファイルや Vector Works ファイルのように「ページや図面という空間とその中に存在するグラフィックオブジェクトの集合」という、ベクトル画像形式に近い構造になっています。したがってワードプロセッサで作成した場合であっても、PDF 中には「本文」といった概念は存在せず、ページのほとんどをカバーするテキストオブジェクトがあるだけです。このことから、PDF 中のテキストを抽出するには各ページについて文章オブジェクト (text object) を検索して、その内容を取り出していく形になります。

2.2. PDF 中のオブジェクト

PDF では次のようなオブジェクトが定義されています。

・Boolean values
　真理値。`/true/` か `/false/` のどちらか。

・Integer and real numbers
　数値を表す。書式は `/¥d/` や `/¥f/` 。

・Strings
　文字列を表す。書式は小カッコ（ `/¥([^¥)]+¥)/` ）や不等号カッコ（ `/<[¥>]+>/` ）を使う。後述。

・Names
　スラッシュで始まる ID。たとえば `/Author` など。

・Arrays

データの列を表す。書式は大カッコで囲い、(`/¥[[^¥]]+¥]/`) 解釈可能なオブジェクトをその中に列挙する。区切りはスペースや改行。

・Dictionaries

値の組を表す。書式は二重の不等号カッコの中に「キー、値」の順に書き連ねる。例を挙げる。

```
<< /Author (YAMAMOTO Suguru) /Age 25 >>
```

これは次のような意味になる。

- Author (Name object) に対応する値が YAMAMOTO Suguru (String object)
- Age (Name object) に対応する値が 25 (Integer object)

・Streams

任意のデータを固まりとして扱える単位にしたオブジェクト。stream という名前が示す通り、全体を単体として扱うのではなく､長さを意識せず先頭から順に最後まで読んでいく (ファイル IO のように)、という使い方をする。

・The null object

「無い」ことを表す。書式は `/null/` あるいは何も書かないことでこれを表す。

・Indirect object

他のオブジェクトから参照可能な「ラベル（数値)」を持つオブジェクト。内容としては、他のオブジェクトを複数内包するのみ。書式は `/¥d ¥d obj/` という行と `/endobj/` という行で内容を囲う。

先に進む前に、文字列の表現について少し補足します。PDF では文字列 (string object) を表す場合､「リテラル文字列 (literal strings) 」および「16 進文字列 (hexadecimal strings) 」を使う二つの方法があります。文字列リテラルは小カッコで囲われたリテラルで、基本的に内容がそのまま文字情報として扱われます（C 言語で使う引用符の文字列リテラルとほぼ同じ)。ただし内容を正しく抽出するにはエスケープシーケンスや対括弧を処理する必要があります。16 進文字列は不等号カッコで囲われたリテラルで､「文字の形状（グリフ)」を示す数値 (ID) を 16 進数で記述するものです。一般的に文字の形状を示す数値といえば文字コードそのものですが、PDF では一般的に使われる文字コードをそのまま使って数値表現しているとは限りません（この点については後の CMap の項で触れます)。詳細なリテラルの記述規則は PDF Reference の 3.2.3 項"String Objects"を参照してください。

2.3. テキストの定義

PDF のデータ構造概要
（PDF Reference 3.4.4項を参照）

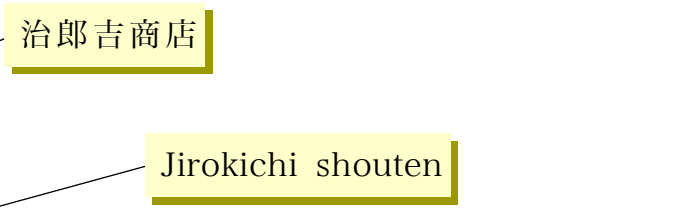
・trailer の /Root キーに対応する value で参照するオブジェクト
  ・/Type/Catalog
    ・/Type/Pages
      ・/Type/Page
        ・ページを構成するオブジェクト群

ほとんどの PDF ファイルではデータを直接記述しません。PDF の概念ではまず Root オブジェクトがあり、その中に Catalog(?)があり、その中に Pages ('Page'の array) があり、その中に Page (object の集合)があるようです。こうした構造になっていることから、ファイルの先頭からグラフィックオブジェクトデータがずらりと並んでいるわけではなく、ページ単位で論理的にデータが構成されています。具体的には、Indirect object を使ってまず Root を構成する Catalog オブジェクトにラベルを付け、Root からそれを参照します。同様に Catalog から Pages を、Pages から object を参照す

る形で記述されることがほとんどのようです。したがって厳密にテキストを抽出しようとすれば、ページ順序を考えなければならないため、しっかりとツリー構造の参照関係をたどっていく必要があります。しかし、検索用インデックスの作成が目的であればページ順序は無視して良いので、ファイルを先頭からオブジェクトをひたすら検索して「そのオブジェクトに含まれるテキストを抽出して次を検索」を繰り返せば目的を達すると言えます。

具体的に、テキストを含む Indirect object（以下、参照オブジェクト）の例を次に示します。

```
2 0 obj
<</Length 193>>
stream
0.1 w
q 0 0 595.3 842 re W* n
q 0 0 0 rg
BT
63.8 777.2 Td /F1 18 Tf<0102030405>Tj
ET
Q
q 0 0 0 rg
BT
106.3 694.7 Td /F2 18 Tf[<01020304>6<05020607>6<0208>-1<0907040A>6<0B>-6<0C>6<0D>]TJ
ET
Q
Q
endstream
endobj
```



この参照オブジェクトは、二重の不等号カッコで囲われた Dictionary オブジェクトと、/stream/という行と/endstream/という行で囲われた Stream オブジェクトの二つで構成されています。なお Stream オブジェクトは直前にそのストリームの長さなどを記述した Dictionary オブジェクトが必要と定められているので、これらは必ずペアで現れます。この参照オブジェクトの内容は Stream オブジェクトで記述されていますので、中を読むと、BT という行と ET という行が見つかります。これらは Begin Text と End Text の略で、この間に（グラフィックオブジェクトとしての）テキストオブジェクト情報が記述されます。詳細な書式については PDF Reference の 5 章を参照してください（膨大な内容ですが）。テキストオブジェクト中にある文字列オブジェクトが、テキストオブジェクトの文字内容を構成しています。ちなみに、この例ではテキストオブジェクトが二つ含まれており、前者が「治郎吉商店」、後者が「Jirokichi shouten」です。不等号カッコで文字情報が示されていますが、0x01 が前者では「治」、後者では「J」となっています。この数値と文字の関連については CMap の項を参照してください。

2.4. ストリームの圧縮

Stream オブジェクトでは、その内容を PDF の書式に乗っ取ってそのまま書く代わりに、内容を圧縮した結果をバイナリデータで埋め込むことができます。圧縮に使えるアルゴリズムには PDF 1.6 で 10 種類あり、無圧縮も可能です。しかしながら圧縮アルゴリズムとしては無償で制約も無く使用でき、かつ圧縮率も高い zlib の deflate 圧縮を使うことが多いようです。

前節で示した例は、stream の内容を圧縮していませんでした。これを圧縮した場合の例を次に示しま

す。

```
2 0 obj
<<
/Length 144
/Filter /FlateDecode
>>
stream
x 彎・ツ0E ・・・i 痘 BX?キ B タ A ワャ C `・セ)yp9¥8 ・シユ,轉ァ[・b 柝 Z 碼妹・・|ォ DD[・ヨ塵漕・・・(略)
ゥ 1・・フ n 頬 HM ノ塲サ t ノヌ彁ト・)F
endstream
endobj
```

圧縮を使用した結果、前回とは stream の内容が変化しています。そのため stream についての情報を記した Dictionary オブジェクトの内容も変化しています。まず stream の長さが 193 から 144 に短くなっています。また、圧縮アルゴリズムを指定する Filter というキーが新しく追加され、これに FlateDecode という値が設定されています。この値は、zlib の deflate アルゴリズムを使っていることを示しています。この他の値については PDF Reference の 3.3 節を参照してください。

2.5. CMap (character map)

一般的に文字の形状と数値の対応関係を表す概念を「文字コード」と呼びますが、PDF では文字情報を表すのに特定の文字コードに依存していません。簡単に言えばファイルごとに独自の文字コード体系を作って使用できるようになっています。PDF におけるこのファイルローカルな文字コードは CMap と呼ばれます。これが具体的にどこで使われるかというと、不等号カッコで囲われる「16 進文字列」で文字形状を指定するのに使われます。もし PDF からテキストを抽出する場合、テキストオブジェクト中に「16 進文字列」を見つけた場合はこれを解釈してまず文字の値を一斉に取得し、その上でそれらの値が、たとえば Unicode におけるどの文字コード値に相当するのかを計算する必要があります。

CMap をテキストオブジェクトに適用する場合の流れを記します。テキストオブジェクトの定義中に /F¥d/ という名前オブジェクトがあると、この名前オブジェクトをキーにした Dictionary オブジェクトを PDF ファイル中から探すようです。Dictionary を見つけたら、そのキーに対応した値（ほとんどの場合は参照オブジェクト）を取り出し、その内容でフォントを適用します。このフォント情報は一つの Dictionary になっており、その中で ToUnicode という名前オブジェクトのキーに対応する値として CMap の定義が指定されます。詳細な情報については PDF Reference 5.9.2 項を参照してください。次に、具体的なテキストオブジェクトとそれをレンダリングするのに必要な CMap 等のオブジェクトを例として挙げます。

```
2 0 obj
（中略）
BT
63.8 777.2 Td /F1 18 Tf<0102030405>Tj
ET
（中略）
BT
106.3 694.7 Td /F2 18 Tf[<01020304>6<05020607>6<0208>-1<0907040A>6<0B>-6<0C>6<0D>]TJ
ET
（中略）
```

```
endobj

15 0 obj
<<
/F1 14 0 R
/F2 9 0 R
>>
endobj

14 0 obj
<<
/Type/Font/Subtype/TrueType/BaseFont/BAAAAA+MS-PGothic
/FirstChar 0
/LastChar 5
/Widths[1000 1000 1000 1000 1000 1000 ]
/FontDescriptor 12 0 R
/ToUnicode 13 0 R
>>
endobj

13 0 obj
<</Length 403>>
stream
/CIDInit/ProcSet findresource begin
12 dict begin
begincmap
/CIDSystemInfo<<
    /Registry (Adobe)
    /Ordering (UCS)
    /Supplement 0
>> def
/CMapName/Adobe-Identity-UCS def
/CMapType 2 def
1 begincodespacerange
<00> <FF>
endcodespacerange
5 beginbfchar
<01> <6CBB> % 6CBB は Unicode で「治」
<02> <90CE> % 90CE は Unicode で「郎」
<03> <5409> % 5409 は Unicode で「吉」
<04> <5546> % 5546 は Unicode で「商」
<05> <5E97> % 5E97 は Unicode で「店」
endbfchar
endcmap
CMapName currentdict /CMap defineresource pop
end
end
endstream
endobj
```

必要な処理

PDF 書式の解析だけではテキストを抽出できません。テキスト抽出に必要な処理は次です。

1. PDF 書式の解析

テキストオブジェクトの抽出と解釈

2. 圧縮されたデータストリームの展開
   zlib が使われる場合がほとんど
3. PDF のテキストデータの抽出
   文字列リテラルの解析（エスケープシーケンスの処理を含む）と 16 進リテラルの解析
4. 16 進リテラルから抽出した CID と、Unicode の文字コード値へのマッピング
   CMap ファイルの解析および適用

以上