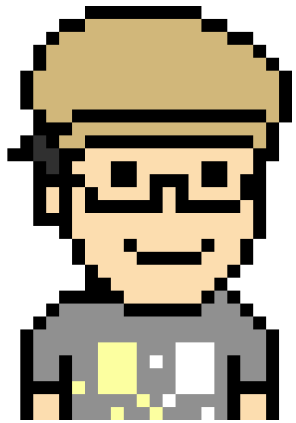
lopdfの話

2018-08-01 Rust LT #2 〜いま使う！Rust〜

小嶋智

自己紹介

- skoji
- プログラマ
- 主にRuby
- テキスト処理や電子出版の周辺を漂っています



話す内容

lopdf

- PDFライブラリ
- 製品で使った話をします

背景

対象の製品

HTML/CSS Layout Engine

VersaType Converter

旧 Vivliostyle Formatter

VersaType ConverterはHTML/CSSからPDFを一括生成するソフトウェアです。CMSなどと連携し、サーバ上で動作します。

VersaType Viewer

旧 Vivliostyle Viewer

VersaType Viewerはブラウザ上で印刷物のページレイアウトを表示するWebアプリケーションです。

https://trim-marks.com/

VersaType

- HTML/CSSをページ組版するエンジン
  - 原則としてCSS標準の範囲
  - JavaScript

VersaType Converter

- HTML/CSSをPDFに変換
- JSの組版エンジン + 組み込み用Chromium
- Win/Linux/macOSのバイナリ

なぜPDF処理(1)

- PDF生成はChromium任せ
- PDF出力をもっと強くしたい
- 後処理 vs Chromiumへのパッチ
  - 向き不向きがある

なぜPDF処理(2)

- Chromiumへのパッチ
  - Chromiumは大きい上に変化が激しい
  - 本家に取り込まれない場合は追従がしんどい
- 後処理でできることは後処理で

PDF後処理: 要件

組み込んで配布

PDFの低レイヤ

Win/Linux/macOS

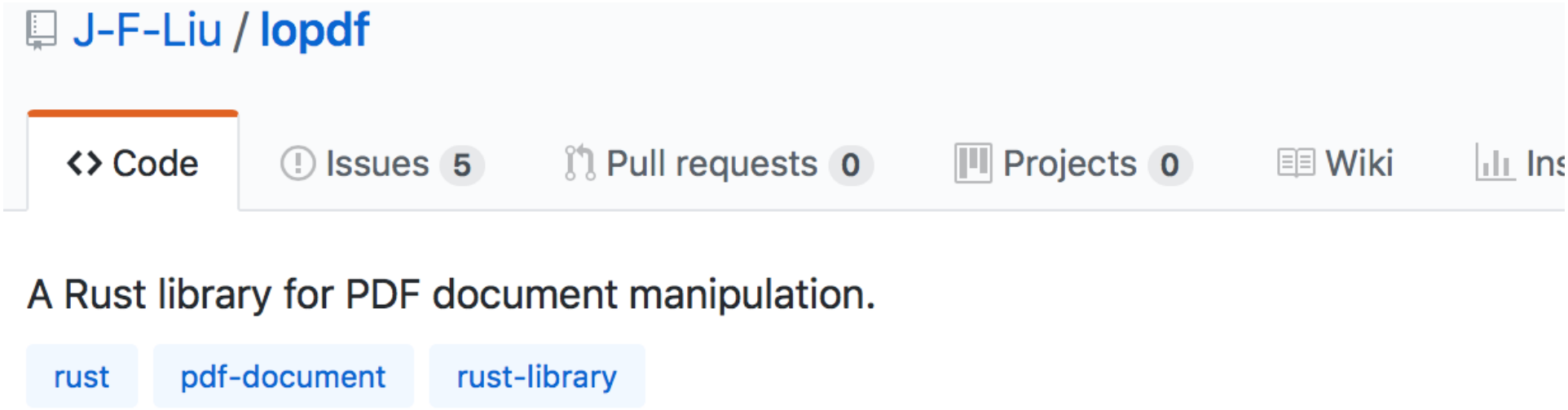
C++から使う

C++書きたくない

（できるだけ）

探した結果

lopdf !



- バイナリに組み込める
- PDFの低レイヤ操作ができる
- Linux/Windows/macOSのx86_64
- C++から呼べる
- Rustで書ける

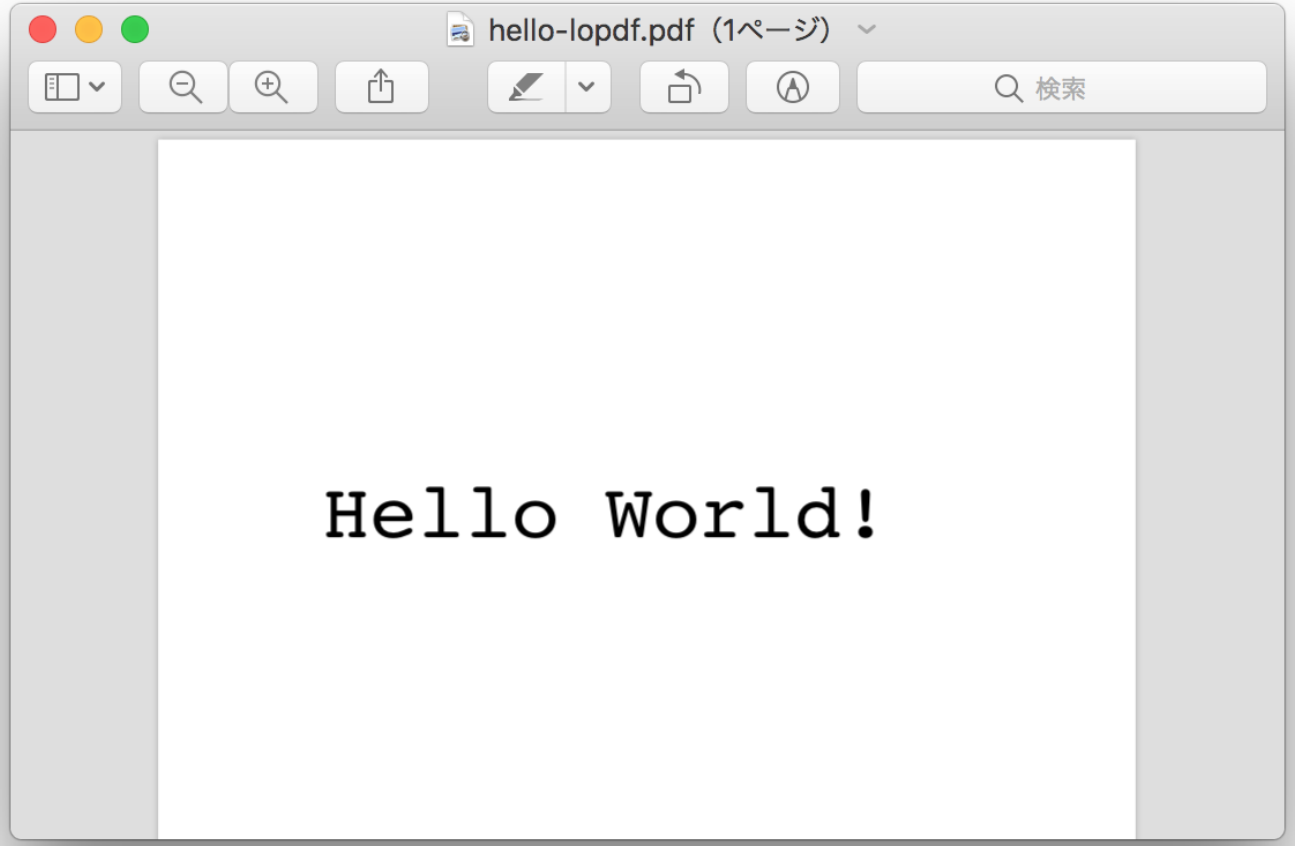
懸念

- PDF操作の機能が本当に足りているか
- C++からの呼び出し
  - CSSのbookmarks指定をPDF Outlinesに反映
  - そこそこ複雑な構造を渡すインタフェース

機能が足りているか

- 結論から言うと、足りている。
- 低レイヤなのでコード量は多い
- RubyのPrawnと比較してみる
  - PDF生成専用
  - 簡単に書けるが低レイヤ操作は難しい

機能比較: Hello World



Prawn

```
require 'prawn'

Prawn::Document.generate('hello.pdf') do
  text "Hello World!"
end
```

Prawn：位置とフォントを指定

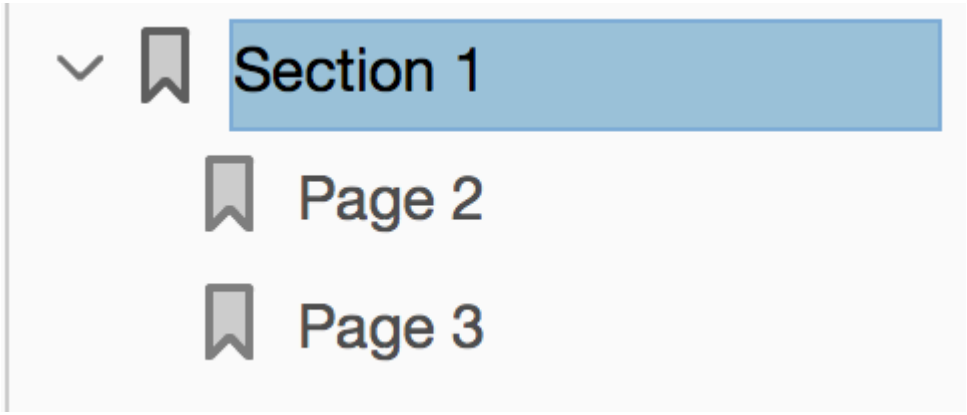
```
require 'prawn'

Prawn::Document.generate('hello2.pdf') do
  font("Courier") do
    font_size 48
    draw_text "Hello World!", :at => [100,600]
  end
end
```

lopdf

```rust
fn hello() {
    let mut doc = Document::with_version("1.5");
    let pages_id = doc.new_object_id();
    let font_id = doc.add_object(dictionary! {
        "Type" => "Font",
        "Subtype" => "Type1",
        "BaseFont" => "Courier",
    });
    let resources_id = doc.add_object(dictionary! {
        "Font" => dictionary! {
            "F1" => font_id,
        },
    });
    let content = Content {
        operations: vec![
            Operation::new("BT", vec![]),
            Operation::new("Tf", vec!["F1".into(), 48.into()]),
            Operation::new("Td", vec![100.into(), 600.into()]),
            Operation::new("Tj", vec![Object::string_literal("Hello World!")]),
            Operation::new("ET", vec![]),
        ],
    };
    let content_id = doc.add_object(Stream::new(dictionary! {}, content.encode().unwrap()));
    let page_id = doc.add_object(dictionary! {
        "Type" => "Page",
        "Parent" => pages_id,
        "Contents" => content_id,
    });
    let pages = dictionary! {
        "Type" => "Pages",
        "Kids" => vec![page_id.into()],
        "Count" => 1,
        "Resources" => resources_id,
        "MediaBox" => vec![0.into(), 0.into(), 595.into(), 842.into()],
    };
    doc.objects.insert(pages_id, Object::Dictionary(pages));
    let catalog_id = doc.add_object(dictionary! {
        "Type" => "Catalog",
        "Pages" => pages_id,
    });
    doc.trailer.set("Root", catalog_id);
    doc.compress();
    doc.save("hello-lopdf.pdf").unwrap();
}
```

機能比較: Outlines



Prawn

```ruby
require 'prawn'
Prawn::Document.generate('outline.pdf') do
  (1..3).each do |index|
    text "Page #{index}"
    start_new_page
  end

  outline.define do
    section('Section 1', destination: 1) do
      page title: 'Page 2', destination: 2
      page title: 'Page 3', destination: 3
    end
  end
```

lopdf

```rust
fn outline() {
    let mut doc = Document::with_version("1.5");
    let pages_id = doc.new_object_id();
    let font_id = doc.add_object(dictionary! {
        "Type" => "Font",
        "Subtype" => "Type1",
        "BaseFont" => "Courier",
    });
    let resources_id = doc.add_object(dictionary! {
        "Font" => dictionary! {
            "F1" => font_id,
        },
    });

    let mut pages_list: Vec<Object> = Vec::new();

    for x in 0..3 {
        let str = format!("Page {}", x + 1);
        let content = Content {
            operations: vec![
                Operation::new("BT", vec![]),
                Operation::new("Tf", vec!["F1".into(), 48.into()]),
                Operation::new("Td", vec![100.into(), 600.into()]),
                Operation::new("Tj", vec![Object::string_literal(str)]),
                Operation::new("ET", vec![]),
            ],
        };
        let content_id = doc.add_object(Stream::new(dictionary! {}, content.encode().unwrap()));
        let page_id = doc.add_object(dictionary! {
            "Type" => "Page",
            "Parent" => pages_id,
            "Contents" => content_id,
        });
        pages_list.push(page_id.into());
    }
    let outline_id = {
        let outline_id = doc.new_object_id();
        let outline_first_id = doc.new_object_id();
        let outline_second_id = doc.new_object_id();
        let outline_third_id = doc.new_object_id();


        let action_first_id = doc.add_object(dictionary!{
            "D" => vec![pages_list[0].clone(), "FitH".into(), Object::Null],
            "S" => "GoTo"
        });
        let action_second_id = doc.add_object(dictionary!{
            "D" => vec![pages_list[1].clone(), "FitH".into(), Object::Null],
            "S" => "GoTo"
```

```
        });
        let action_third_id = doc.add_object(dictionary!{
            "D" => vec![pages_list[2].clone(), "FitH".into(), Object::Null],
            "S" => "GoTo"
        });

        let outline_second = dictionary! {
            "Title" => Object::string_literal("Page 2"),
            "Parent" => outline_first_id,
            "A" => action_second_id,
            "Next" => outline_third_id
        };
        let outline_third = dictionary! {
            "Title" => Object::string_literal("Page 3"),
            "Parent" => outline_first_id,
            "A" => action_third_id,
            "Prev" => outline_second_id
        };
        let outline_first = dictionary! {
            "Title" => Object::string_literal("Section 1"),
            "Parent" => outline_id,
            "A" => action_first_id,
            "First" => outline_second_id,
            "Last" => outline_third_id
        };
        let outline = dictionary! {
            "Count" => 2,
            "First" => outline_first_id,
            "Last" => outline_first_id,
        };
        doc.objects.insert(outline_first_id, Object::Dictionary(outline_first));
        doc.objects.insert(outline_second_id, Object::Dictionary(outline_second));
        doc.objects.insert(outline_third_id, Object::Dictionary(outline_third));
        doc.objects.insert(outline_id, Object::Dictionary(outline));
        outline_id
    };
    let pages = dictionary! {
        "Type" => "Pages",
        "Kids" => pages_list,
        "Count" => 3,
        "Resources" => resources_id,
        "MediaBox" => vec![0.into(), 0.into(), 595.into(), 842.into()],
    };
    doc.objects.insert(pages_id, Object::Dictionary(pages));
    let catalog_id = doc.add_object(dictionary! {
        "Type" => "Catalog",
        "Pages" => pages_id,
        "Outlines" => outline_id
    });
    doc.trailer.set("Root", catalog_id);
    doc.compress();
    doc.save("outline.pdf").unwrap();
}
```

なげえ

とはいえ

lopdfは低レイヤを触れる

- PDF outlinesのアクションはなんでもあり。
  - Prawnでは固定。
- コードの短さは優先度低い

```
let action_first_id = doc.add_object(dictionary!{
    "D" => vec![pages_list[0].clone(),
                "FitH".into(),
                Object::Null],
    "S" => "GoTo"
});
```

C++から呼び出し

Bookmarksのデータ

- ラベル、階層、ページ番号
  - の配列

```
[{ title: '第１章', level: 1, page :1 },
 { title: '第１章第１節', level: 2, page: 1 },
 { title: '第１章第２節', level: 2, page: 9 },
 { title: '第２章', level:1, page: 20 }]
```

最初の案

- Rust側で空のvectorを生成
- C++からtitle, level, pageの組をvectorにひとつずつ追加
- C++からPDFのパスを渡してoutlines書き込み処理

インタフェース

```
void *new_vector();
bool add_outline_to_vector(const char *title,
                           int_32t level,
                           int_32t page,
                           void *vector);
bool write_outline(const char *src_path,
                   const char *dst_path,
                   void *vector);
```

処理の流れ

- JavaScriptでbookmarks情報作成
  - オブジェクトの配列
- C++のデータ構造に変換
- Rustから出ているAPIを順次呼ぶ

だるい

欠点

- C++コード上で目次の構造を作る必要がある
  - JSとRustの間に入るC++のコードでは、本来知らなくても良い知識

2つ目の案

- JS側で`JSON.stringify()`
- C++では文字列の中身に関知しない
- Rust側ではパースしてVectorを作る
  - `serde-json`

インタフェース

```
bool add_outline(const char *src_path,
                 const char *dst_path,
                 const char *outline_json);
```

今回はこれを採用

デモ

cssの指定

```
h1 {
  bookmark-level: 1;
}
h2 {
  bookmark-level: 2;
}
h3 {
  bookmark-level: 3;
}
.cover-page h1 {
  bookmark-label: content(text) " 表紙";
}
```

この資料のPDF outlinesはこの指定で生成した。

リソース

- lopdf
  - https://github.com/J-F-Liu/lopdf
- Prawn
  - http://prawnpdf.org/
- Prawnとlopdf比較のソース
  - https://github.com/skoji/compare-prawn-lopdf
- このスライドのソース
  - https://github.com/skoji/rust_lt_20180801
- VersaType
  - https://trim-marks.com