情報特別演習 最終レポート
「TeX コンパイルサーバの作成」

学籍番号 200911434
名前 青木大祐

2013 年 1 月 22 日

1 背景

文書や論文作成用のツールとして、広く用いられているソフトウェアのひとつに TeX が挙げられる。このツールはテキストベースで綺麗な書類や数式を記述でき、またプログラミング言語のようにマクロを記述できるなど高い拡張性を持っている。しかし、この TeX には幾つかの問題がある。

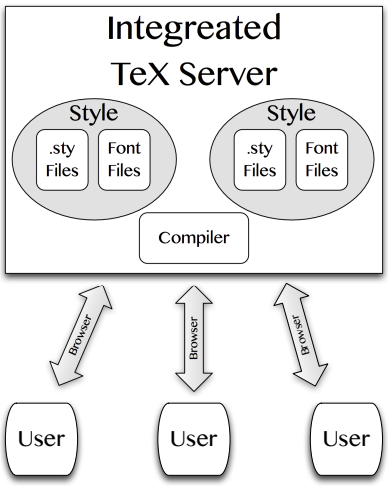
まず、環境構築の手間が大きい点が挙げられる。例えば MacOS X に TeX を導入するための Mac TeX[*1] パッケージのサイズは 2GB を超えており、ダウンロードするだけで非常に長い時間がかかる。また、複数のコンピュータでスタイル (.sty ファイルやフォント) を共有する際に問題が起きることがあり、文書を作成したコンピュータではコンパイルが出来たはずなのに、他のコンピュータでコンパイルしようとした時にスタイルが適用できなくて失敗することがある。

次に、文法が難解であることも問題である。標準で用意されている機能を使っているうちは良いが、自分でマクロを記述しようとすると、複雑な TeX 言語の仕組みについての学習が必須となる。また、箇条書き—itemize など高頻度に使用する命令であっても記述量がかなり多く、ワープロソフトを用いればボタンひとつで完了できる操作であっても、TeX 言語では幾つもの命令を入力する必要がある。

こうした問題を抱えている TeX であるが、普段から利用しているソフトウェアだけに、代替品を探すというのも難しい。より TeX を使いやすくするために、これらの問題点のうち、幾つかの点について改善を試みた。

2 方針

まず、環境構築についてのコストを軽減するために、実際に動作する TeX のシステムを 1 台のサーバに集約することを考える。これによって、スタイルファイルの共有についても同じく解決されることが期待される。サーバにはブラウザからアクセスできる Web UI を実装し、TeX がインストールされていないコンピュータからはブラウザを介して TeX の機能を利用できるようにする。



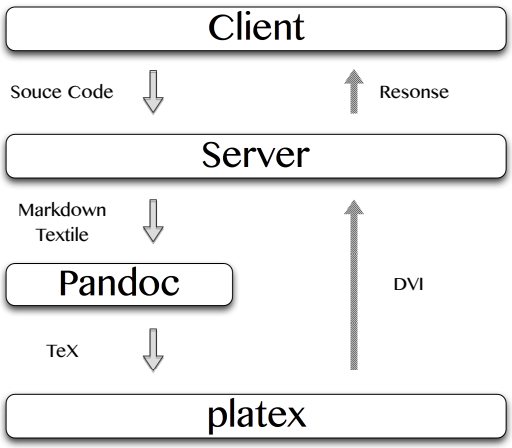
次に、TeX 言語の記述量の多さを軽減するために、簡略な表記の導入を考える。これは TeX 言語のレイヤでマクロを実装するのでは無く、TeX に入力を与える前段階でのプリプロセスとして TeX 言語に変換する。TeX 言語のように構造を持ったテキストベースの記法のうち、メジャーである Markdown 記法と Textile 記法の 2 つを、代替として、また TeX 言語と混在して記述できるようにする。

[*1] http://tug.org/mactex/

3　実装

前章の方針を基に、問題点を解決するためのプログラムを実装した。このプログラムは大きく分けて 2 つの部分に分かれており、サービスを提供するコンピュータで動作する部分を**サーバ**、サービスを利用するコンピュータで動作する部分を**クライアント**と呼ぶことにする。

以下に、システムの概要を図示する。



制作に際して、サーバ側にはスクリプト言語 Perl を、クライアント側には HTML5+JavaScript を用いて実装を行った。

3.1　サーバ

サーバ部分の処理は、次のような流れで行われる。

1. ブラウザからのリクエストに応答して Web UI(クライアント) を返す
2. クライアントに入力されたソースコードを Ajax で受け取る
3. 簡略表記が使われていれば、ルールに従って TeX 言語に変換する
4. ソースコードを TeX に渡し、DVI を生成する
5. 生成した DVI をクライアントに返す
6. ダウンロード要求を受け取り、PDF を生成して返す

これらを実現するために、スクリプト言語 Perl で用いられる軽量 WAF[*2]である Amon2[*3]を用いて実装を行った。Amon2 は PSGI アプリケーションを記述するための軽量フレームワークであり、少ない記述量で高機能かつ安全なアプリケーションが制作できる。同じく Perl の WAF として広く用いられている Catalyst に対してシンプルな仕組みが特色であり、学習にかかるコストも比較的小さい点も魅力である。

以下に、各ステップごとの具体的な処理の内容を説明する。

[*2] Web Application Framework
[*3] http://amon.64p.org/

3.1.1 リクエストへの応答

ユーザが TeX 言語などのソースコードを入力する画面を Web サーバとして提供する。クライアントの詳しい実装については次節で述べる。

3.1.2 ソースコードを受信

Ajax でソースコードを受信するための URL で POST を受け取り、ファイルとして保存する。保存先のディレクトリは他のファイルと衝突しないように一意な名前が振られるようになっており、URL を保存しておくことで任意のタイミングでファイルが参照できる。

3.1.3 形式の変換

簡略表記として Markdown 記法または Textile 記法が使われている場合は、これを TeX 言語に変換する。全体が簡略表記で記述されている場合は、ドキュメント変換ツールの Pandoc[*4]を用いて変換を行う。

また、TeX 言語に埋め込んで簡略表記が使われている場合は Pandoc が使えないため、自作の変換ツールを用いて変換を行う。これは TeX 言語中の任意の場所で簡略表記が使える代わりに、Pandoc を用いるパターンよりも機能が少なくなっている。

3.1.4 DVI の生成

生成された TeX 言語のソースファイルを、TeX の処理系のひとつである platex を用いて DVI に変換する。TeX の命令として OS のコマンドを実行するものがあり、そのままでは危険なので予め使えないように設定しておく必要がある。

3.1.5 DVI をクライアントに返す

送られてきたリクエストへのレスポンスとして、生成された DVI を返す。この段階で PDF を生成しない理由としては、PDF よりも DVI のほうがファイルサイズが小さいため、送信にかかる時間が節約できることが挙げられる。

3.1.6 PDF のダウンロード

ダウンロード用の URL にリクエストを受け取ると、dvipdfmx を用いて PDF を生成し、これをユーザに返す。

3.2 クライアント

次に、クライアントの実装について述べる。

クライアントはブラウザで表示するため、HTML5 を用いて実装している。ページデザインには、Amon2 にバンドルされている CSS ライブラリ、Twitter Bootstrap[*5] を使用している。

クライアント側の処理の流れは以下のようになる。

1. ページを表示する
2. ユーザの入力を Ajax でサーバに送信する
3. 帰ってきた DVI を表示する
4. 内容に問題が無ければ PDF をダウンロードする

以下に、各ステップごとの具体的な処理の内容を説明する。

[*4] http://johnmacfarlane.net/pandoc/
[*5] http://twitter.github.com/bootstrap/

3.2.1 ページを表示する

サーバからページを受信し、ブラウザに表示する。具体的なページのレイアウトに関しては、後述する。

3.2.2 入力を送信

用意された送信用の URL に対して、フォームの内容などを非同期通信で POST する。

3.2.3 DVI の表示

サーバから返ってきた DVI を、dvi.js[*6]を用いてページ内にインライン表示する。実装当初は生成したサムネイル画像を表示する設計であったが、ファイルサイズの問題からどうしても受信に時間が掛かってしまうため、JavaScript で DVI を表示する dvi.js を採用することにした。この場合、受信から描画に掛かる時間は、もはやユーザがストレスとして感じない程度に削減することができる。また、ファイルサイズとの兼ね合いで低解像度で表示せざるを得ない画像と比べて、ブラウザに表示されるページの一部として描画されるため、dvi.js を用いて描画した場合のほうが綺麗に表示されるという利点もある。

実際に dvi.js によって DVI が描画されているスクリーンショットを以下に示す。ページ中央のテキストボックスにソースコードを入力し、下部のコンパイルボタンをクリックすることで、最下部に DVI プレビューが表示される。

3.2.4 PDF のダウンロード

ダウンロード用の URL にアクセスすることで、PDF をダウンロードできる。

[*6] https://github.com/naoyat/dvi.js/wiki

以上が、サーバとクライアントの実装の概要である。

4　考察

これらの機能を実装することで、TeX の環境構築に掛かる時間的コストは削減することが可能になったと考える。また、簡略表記の導入によって学習コストを軽減し、非利用者に対する入門の敷居を下げることにも繋がると思う。

5　今後の課題

今回 TeX コンパイルサーバを実装して、足りないと思った機能がいくつかあり、これらは今後の課題として逐次実装していけたら良いなと思っている。

まず、ユーザ個人用のファイルを保存する機能が必要である。講義の課題などのシンプルな書類であれば問題ないこともあるが、この報告書のようなレポートや論文などにおいて、画像が挿入できないのは致命的である。

また、エディタが貧弱なのも課題といえる。これは命令の挿入など入力を支援する機能や、シンタックスハイライトをクライアント側に実装することで解決されると考えられる。

最後に、私のように普段から TeX を使うユーザにとっては、コマンドラインから操作できたほうが都合の良いことは多い。そのため、ブラウザだけでなくコマンドラインからのリクエストを送信できるようなソフトウェアも作成したいと思う。

6　感想

もともと私は Perl を主に用いていたのだが、今まで Web アプリケーションを書いたことは無かった。しかし CGI で一大勢力を築いただけあって、Perl の Web における強みは周知の事実である。折角 Perl を使っていながら Web アプリケーションを書かないというのはもったいないという事で、今回の演習には Perl を用いて実装を行なってみた。その結果、Perl という言語の柔軟性、また拡張性の高いフレームワークの強力さが実感できたように思う。

また、クライアント部分に用いた JavaScript は私の苦手な言語の筆頭であった。しかし、ある程度の量を書くと、一見奇妙に思える言語の特性が、ブラウザ上で動作する言語として大きな強みになっている部分もあるのだと分かった。この演習を通して苦手意識が薄れたのは大きな収穫だったと思う。

なお、今回の演習で作成したソフトウェアは Github にて公開[*7]している。

[*7] https://github.com/VienosNotes/DominoTeX