Bikeshed.rs

Basic Information

- Name: 💭
- Email: 💭
- GitHub: github.com/whichxjy
- Website: whichxjy.com
- Time Zone: GMT +8

Project Abstract

The goal of this project is to offer a faithful rewrite of bikeshed, a tool used for generating many standards, in Rust.

Project Description

➢ Motivation

▹ Why Rust?

Now `bikeshed` is written in **Python**, which is one of the world's most popular programming languages. It's most praised for its elegant syntax and readable code, but it's also known for being slow in some cases, such as doing large string processing.

Rust is a systems programming language that aims to offer both performance and safety. Usually, memory safety in programming languages like Python comes with a cost of a garbage collector, but Rust solves those issues at compile time with its ingenious type system, allowing for better performance compared to Python.

Criteria	Rust	Python
Performance	😄	😐
Memory Efficiency	😄	😐
Ecosystem	🙂	🙂
Simplicity	😐	😄

▹ Why not extend Python with Rust?

To speed up `bikeshed`, rewriting some of its key components in Rust and importing the Rust-implemented module with rust-cpython or pyo3 is a choice. In my opinion, that may not be a good idea because it would make the project complicated and hard to maintain. What's more, there're many components related to the key components. Moving the key components out means editing many other components in the Python version.

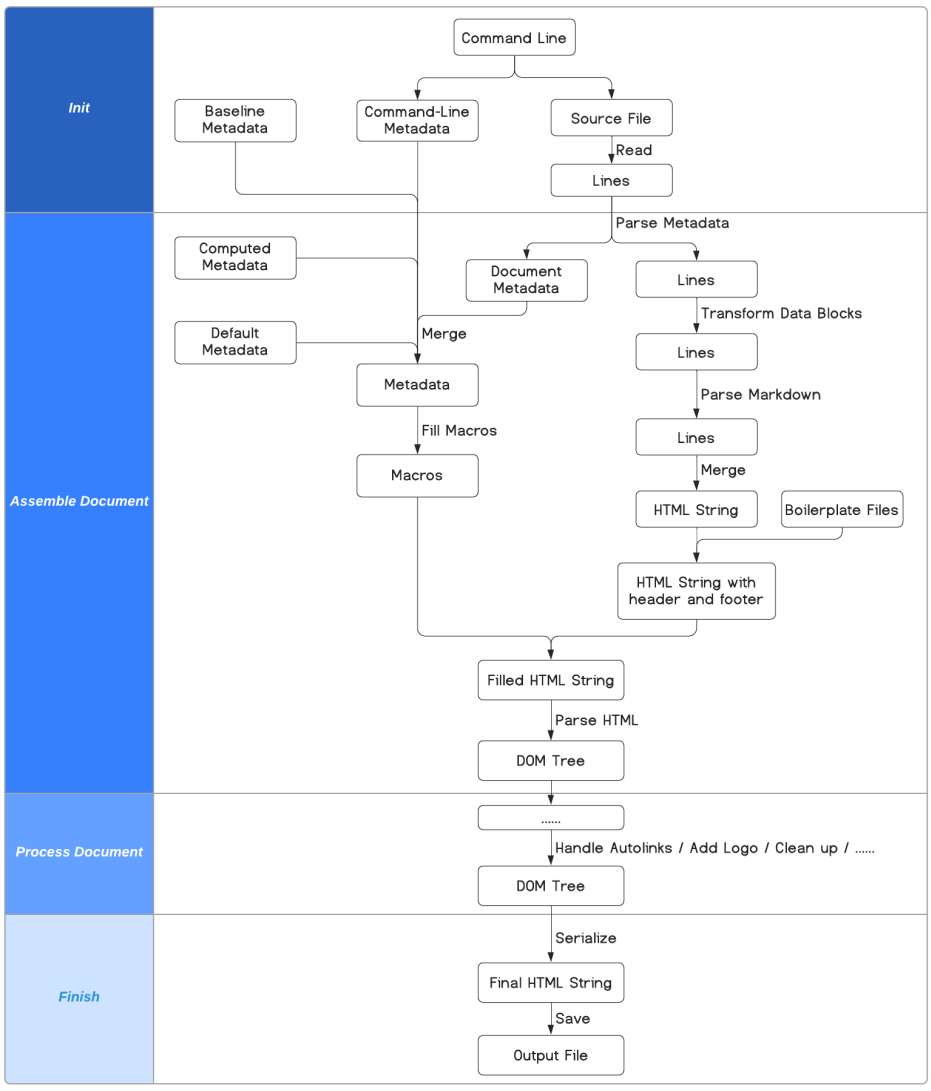
➢ Prototype

I have built a prototype for this proposal: https://github.com/whichxjy/bikeshed-rs-demo. It has some basic features of `bikeshed spec`, such as metadata parsing, macros replacing and document generating, but it's incomplete. When the input file of `spec` command is hello.bs, the output file would be hello.html.

➢ Implementation

▹ `spec` command

The most commonly-used command of `bikeshed` is `spec`. It turns a Bikeshed source file into an output HTML file.



The `Spec` structure is the most important component in `bikeshed`. It implements four main procedures:

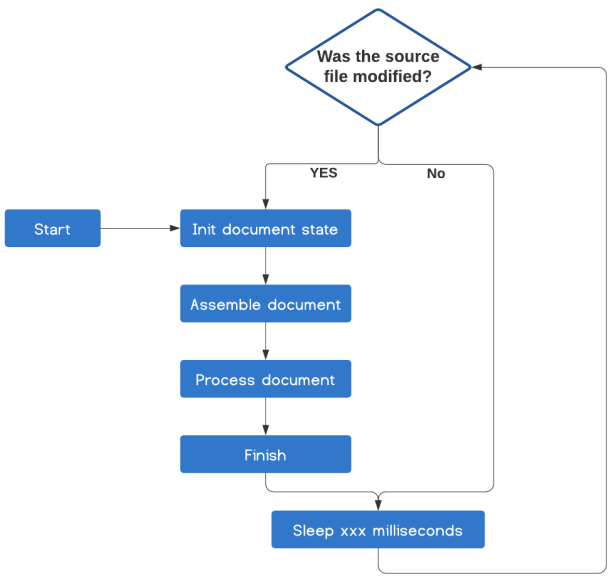
1. **Init**
    - Read the source file into an array of lines.
    - Initialize states of the document.
2. **Assemble document**:
    - Handle metadata / data blocks / markdown / ……
    - Turn lines into a HTML String.
    - Fill the macros in the HTML String.
    - Turn the HTML string into a DOM tree.
3. **Process document**:
    - Process the DOM tree: handle autolinks / add logo / clean up / ……
4. **Finish**:
    - Serialize the DOM tree and save the result as the output file.

▹ HTML manipulation

When it comes to parsing HTML, using html5ever is a good choice. However, `html5ever` does not provide any DOM tree representation. It uses rcdom (a very minimal library) for testing, but we need more than that. Therefore, I would use kuchiki for HTML manipulation. There are some examples in the prototype.

▹ `watch` command

The `watch` command is another common command. It's identical to the `spec` command, except it sets up a watcher and auto-rebuilds the spec every time it changes. To implement this command, we need to create a daemon thread and keep watch on the source file. When the source file changes, we do exactly what we do in the `spec` command: `Init` -> `Assemble` -> `Process` -> `Finish`.



Development Process

➢ May 4 - June 1 (Community Bonding Period)

- Learn more about the Mozilla community.
- Discuss the architecture or other ideas about `bikeshed-rs` with the mentors.
- Build and optimize the CI process for this project.

➢ June 1 - July 3 (Coding period 1)

- Set up tests for `bikeshed-rs`.
- Implement metadata.
- Implement markup shortcuts.
- Implement definitions.
- Implement a part of autolinking.

➢ July 3 - July 31 (Coding period 2)

- Implement the rest of autolinking.
- Implement bibliography.
- Implement boilerplate generation.
- Implement IDL processing.
- Implement testing integration with WPT.
- Discuss it with mentors on whether the `bikeshed spec` command has met the requirements, both functional and non-functional ones.

➢ July 31 - August 24 (Coding period 3)

- Implement bikeshed watch.
- Implement bikeshed template.
- Implement bikeshed echidna
- Implement bikeshed update.
- Implement bikeshed source.
- Implement bikeshed issues-list.
- Clean up the code and remove bugs (if any).
- Write document.

➢ Future Improvements

- Implement a REST API for `bikeshed-rs` so that users can use it without installing.
- Build a Docker container image for `bikeshed-rs` and push it to Docker Hub.