LOSE THE EVIL CONTEXT

C

MOTIVATION

Currently, `mato` parses markdown formatted files with a parser into an abstract syntax tree (AST). For example, the following little file:

```
This is a sentence^(And this is a footnote.)
that is quite long.
```

is parsed into an AST, like the following:

```
Document(DEFAULT,
  Cat(
    Cat(
      Cat(
        Literal("This is a sentence"),
        Footnote(
          Literal("And this is a footnote."))),
      Literal(" that is quite long.")),
    LineBreak))
```

But, not all information that is necessary to render a `groff` file is present in this AST. What is missing is outlined in the following paragraphs.

Preamble is missing from AST

Every `groff` file that is generated by `mato` has a *preamble*, i.e., a default section of `groff` macros, that defines a certain default style. The first few lines of the default preamble look like the following:

```
.PRINTSTYLE TYPESET
.PAGE 5.8i 9i 1i 1i .75i .75i
.PAPER A5
.FAMILY Minion
.PT_SIZE 11
.AUTOLEAD 2
```

This preamble is basically a character string, that is being read in during startup of `matopdf`[1] and spat out as is during rendering at the beginning of the document.

As it is not part of AST, as shown above, it has to be communicated to the renderer in some other way. That other way is a **context** object. And that context object is then being consulted in the renderer.

We can see the code location, where the storing of the preamble happens in the constructor of an AST processor called `MetaDataExtractor`:

```
pub fn new(preamble: &str) -> Box<dyn Process> {
    let mut map = HashMap::new();
    let cp = preamble.to_string();
    if !cp.is_empty() {
        map.insert("preamble".to_string(),
          preamble.to_string());
    }
    Box::new(MetaDataExtractor {
      ctx: map, doc_type: "".to_owned() })
}
```

Here, we see, that a new `HashMap` is being created and the preamble is stored in there under a key called `preamble`.

An interesting aside, that already is hinting at the problem, is the last line of the constructor, where a new object, or

[1] Or even during compile time.

struct of type `MetaDataExtractor` is created and the context object is being put in.

And we can find the location where said preamble is looked up again in the renderer:

```
if self.ctx.contains_key("preamble") {
    let value = self.ctx.get("preamble").unwrap();
    result = format!("{}\n{}\n", result, value);
    self.ctx.remove("preamble");
}
```

PROBLEM STATEMENT AND ANALYSIS

Problem statement

I claim that *context objects are a bad thing*.

But, why, you probably ask. Aren't they used almost everywhere in software development? Aren't they a vital part of many software architectures?