Building a template engine from scratch

Vipul

@vipulbhj

shorturl.at/iyGTZ

DX and Tooling

NativeBase

What exactly is a template engine

Honestly, I have no clue how to define it in a meaningful way, which won’t sound like something I copied from Wikipedia.

So, we will do some supervised learning, for the computer in you called your BRAIN and train it, so it can find the answer, all by itself.

MACHINE LEARNING

This is a template engine

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Try EJS online at: https://ionicabizau.github.io/ejs-playground/.

Basic usage

```
let template = ejs.compile(str, options);
template(data);
// => Rendered HTML string

ejs.render(str, data, options);
// => Rendered HTML string

ejs.renderFile(filename, data, options, function(err, str){
    // str => Rendered HTML string
});
```

This too is a template engine

Installing

Install and update using pip:

```
$ pip install -U Jinja2
```

In A Nutshell

```
{% extends "base.html" %}
{% block title %}Members{% endblock %}
{% block content %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

This, umm.. is also a template engine(sort of)

LEARN REACT › DESCRIBING THE UI ›

Writing Markup with JSX

JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file. Although there are other ways to write components, most React developers prefer the conciseness of JSX, and most codebases use it.

You will learn

- Why React mixes markup with rendering logic
- How JSX is different from HTML
- How to display information with JSX

You get the idea.

So, how do I make a template engine ?

The first step is to define constructs for our template language. This is important, we want our template language to be intuitive, provide a good DX and not be a nightmare to parse, conflicting with out host language, which in this case is HTML.

So, let’s do that.

Our template lang spec

- Comments -> {# comment any text #}
- Value injection -> {{ <variable_name> }}
- loop -> {% for item in items %} … {% endfor %}
- conditional ->

```
{% if predicate_1? %}
{% elif predicate_2? %}
{% else %}
{% endif %}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello, JSWorldConf!</title>
  </head>
  <body>
    <h1>Welcome, {{ user_name }}!!</h1>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello, JSWorldConf!</title>
  </head>
  <body>
    {% if isAdmin %}
      <p>You are Admin</p>
    {% elif isModerator %}
      <p>You are Moderator</p>
    {% else %}
      <p>You are viewer</p>
    {% endif %}
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello, JSWorldConf!</title>
  </head>
  <body>
    <ul>
      {% for product in products %}
        <li>{{product.name}}: {{ product.price }}</li>
      {% endfor %}
    </ul>
  </body>
</html>
```

Parsing / Tokenizing

The goal here is to parse our template file and break it down into tokens of different type.

Thanks to the simplicity of our template language, this is trivial.

Yup, that’s it.

index.js

```
tokenize() {
  /*
   * {{ <expression> }}
   * {% for | if | elif | else | endfor | endif %}
   * {# COMMENT #}
   */
  const re = new RegExp(/({{.*?}}|{%.*?%}|{#.*?#})/, "igm");
  return this.templateStr.split(re);
}
```

```
<body><h1>Hello {{ user_name }}!!</h1></body>
```

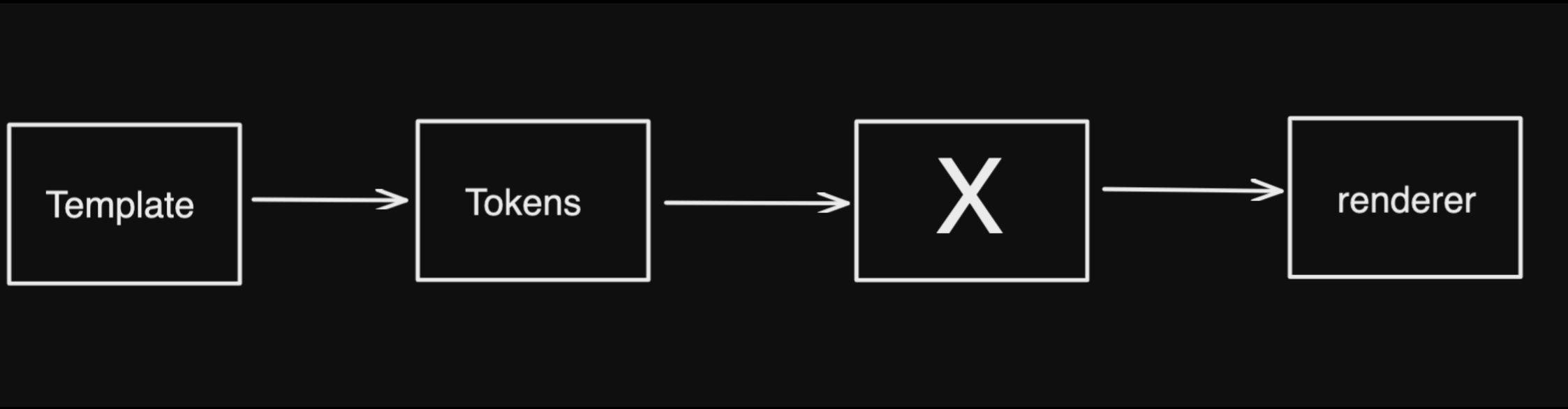
```
[
    '<body><h1> Hello ',
    '{{ user_name }}',
    ' !!</h1></body>'
]
```

```
<p>Welcome, {{user_name}}!</p>
{% if isAdmin %}
  <p>You are Admin.</p>
{% elif isModerator %}
  <p> You are Moderator</p>
{% else %}
  <p>You are not Admin.</p>
{% endif %}

<p>Products:</p>
<ul>
{% for product in product_list %}
  <li>{{ product.name }}: {{ product.price }}</li>
{% endfor %}
</ul>
```

```
[
  '\n<p>Welcome, ',
  '{{user_name}}',
  '!</p>\n',
  '{% if isAdmin %}',
  '\n  <p>You are Admin.</p>\n',
  '{% elif isModerator %}',
  '\n  <p> You are Moderator</p>\n',
  '{% else %}',
  '\n  <p>You are not Admin.</p>\n',
  '{% endif %}',
  '\n\n<p>Products:</p>\n<ul>\n',
  '{% for product in product_list %}',
  '\n  <li>',
  '{{ product.name }}',
  ': ',
  '{{ product.price }}',
  '</li>\n',
  '{% endfor %}',
  '\n</ul>\n'
]
```

Black Box of Magic



```
1 const html = template.render({greeting: "Hello World"});
```

Tokens in Javascript

{# <comment> #} => Ignored while transpiring

{{ <expr> }} => <expr>

{% if predicate? %} => if(predicate?) { …

{% endif %} => }

{% for item in items %} => for(item of items) { ….

{% endfor %} => }

```
function compile(str) {
  const tmpl = [];
  const tokens = tokenize(str);
  for (const token of tokens) {
    if (token.startsWith("{#")) {
      // Comment
      continue;
    } else if (token.startsWith("{{")) {
      const expr = token.substring(2, token.length - 2).trim();
      tmpl.push(expr);
    } else if (token.startsWith("{%")) {
      const expr = token.substring(2, token.length - 2).trim();
      const keywords = expr.split(" ");
      const conditionalConstruct = keywords[0];

      switch (conditionalConstruct) {
        case "if":
          tmpl.push(`if(${keywords[1]}) {`);
          break;
        case "for":
          tmpl.push(`for(const ${keywords[1]} of ${keywords[3]}) {`);
          break;
        case "endif":
        case "endfor":
          tmpl.push("}");
          break;
      }
    } else {
      // Literals
      tmpl.push(token);
    }
  }
}
```

```
function compile(str) {
  const tmpl = [];
  const tokens = tokenize(str);
  for(const token of tokens) {
    if(token.startWith("{#")) {
      // Comment
      continue;
    } else {
      // Literals
      tmpl.push(token);
    }
  }
}
```

```
1  ...
2  ...
3
4
5      else if(token.startsWith("{{")) {
6        const expr = token.substring(2, token.length - 2).trim();
7        tmpl.push(expr);
8      }
9
10
11 ...
12 ...
```

```
...

    else if (token.startsWith("{%")) {
      const expr = token.substring(2, token.length - 2).trim();
      const keywords = expr.split(" ");
      const conditionalConstruct = keywords[0];

      switch (conditionalConstruct) {
        case "if":
          tmpl.push(`if(${keywords[1]}) {`);
          break;
        case "for":
          tmpl.push(`for(const ${keywords[1]} of ${keywords[3]}) {`);
          break;
        case "endif":
        case "endfor":
          tmpl.push("}");
          break;
      }
    }

...
```

```
[
  '\n<p>Welcome, ',
  'user_name',
  '!</p>\n\n',
  'if(isAdmin) {',
  '\n<p>You are Admin</p>\n',
  '}',
  '\n\n<ul>\n',
  'for(const product of products) {',
  '\n<li>',
  'product.name',
  '</li>\n',
  '}',
  '\n</ul>\n'
]
```

The bit that makes it tick

Demo

FIN