Testing?

Aaron Power

20-10-2015

Contents

List of Figures

Chapter 1

Background Research

1.1 Introduction

A transcompiler is a compiler that converts source code into some other form of source code, as opposed to a traditional compiler which converts human readable source code into machine instructions. This has a wide array of uses within different implementations. CMS'(Content Management Systems) have templates that writers pass in their writing to format it into a HTML article on their Website. Academics use LaTeX to convert plain text into scientific documents, and reports. Their use in this thesis is to create a templating language which is a programming language that will convert it's source code into HTML for use in Web applications. The purpose of {NAME TBD} is to create a templating language that Web developers can use to have more modular, and scalable markup in their Web application.

While there is no definitive first templating language. Their popularity began around the late 1990's with XSLT(Extensible Stylesheet Language Transformations) in 1999 in this recommendation (W3C and James 1999). Which was designed for transforming XML(Extensible Markup Language) documents into other formats IE: PDF's as shown in (*Transformation*).

One of the most popular modern templating languages for use within Web applications is (*Mustache*). Mustache is a "*logic-less*" templating language, meaning it has no explicit control flow statements *if, else, for* according to the (*Mustache Manual 5*). Everything is instead based on the object that is passed to the template See Figure 1.1. So if the developer couldn't change the markup if someone's name started with A instead of B unless they add a key to the hash before they pass it to the Mustache templating engine. Mustache is not a HTML specific templating language. Meaning it's syntax is completely independent from it and can be used with anything. However this also means it as language can't use assumptions based on the language leading the syntax to be very verbose.

```
A typical Mustache template:

    Hello {{name}}
    You have just won {{value}} dollars!
    {{#in_ca}}
    Well, {{taxed_value}} dollars, after taxes.
    {{/in_ca}}

Given the following hash:

    {
      "name": "Chris",
      "value": 10000,
      "taxed_value": 10000 - (10000 * 0.4),
      "in_ca": true
    }

Will produce the following:

    Hello Chris
    You have just won 10000 dollars!
    Well, 6000.0 dollars, after taxes.
```

Figure 1.1: A Mustache Example, from (*Mustache Manual 5*)

1.2 Advantages of a templating language in the Web.

As the Web has advanced there has always been a need to be able to have the document delivered to the user to be able to be changed based on who, and how the document is being requested. The most popular way to do this currently is to serve the user a nearly blank HTML(HyperText Markup Language) document containing links to JavaScript files that once downloaded, and executed will fill the DOM(Document Object Model) with the data, and information based on the user's request. Prime examples of this is JavaScript frameworks like React. In which the standard is to have a HTML document with a single <div> element, and let React build the DOM once it can.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Hello React!</title>
        <script src="build/react.js"></script>
        <script src="build/react-dom.js"></script>
        <script src="build/script.js"</script>
    </head>
    <body>
        <div id="example"></div>
    </body>
</html>
```

Figure 1.2: Standard React HTML index file

This comes with problems. The content of the Web page is dependent on the JavaScript. While the JavaScript is being downloaded the user will only see a blank page. This problem becomes especially apparent when the user is on a mobile network, or has a slow connection. Studies have shown that 47% of mobile users expect a site to load in less than two seconds, that 40% of mobile users will abandon a site if it doesn't load in three seconds or less, and that 52% of users said that a quick page load is important to site loyalty according to (*How Loading Time Affects Your Bottom Line*). In large scale applications, and businesses JavaScript size quickly becomes a huge bottleneck in the load times of Websites as shown in (Les 2015).

It is in this area in particular that the advantages of using a templating language is seen. With a templating language the content of the Web page can be sent in the initial HTML page, making the "perceived" load time of the Web page very quick. Giving the JavaScript more time download than if the user were to see a blank page.

Having a templating language can also provide good separation between the data model of the language, and the design of the page. The designers, or client-side programmers who are formatting the Web page don't have to worry about how the server-side logic works, or write code that could change server's data-model in order to change the format of the Web page.

1.3 Main features of {NAME TBD}

From doing my own research and looking at the two comparisons of both templating languages in established server-side programming languages(*PHP, C#, .NET*) from (*Comparison of web template engines*), and templating languages made specifically for use in Isomorphic JavaScript which are generally much younger in their development than their traditional language equivalents as shown in (Alex 2014).

From looking at these languages I think the main features of {NAME TBD} is to first have inheritance, and includes, and variables which is to mean that you can write parent markup files which have child markup files which when parsed will include the markup of the parent, that arbitrary {NAME TBD} files can be inserted into other {NAME TBD} files, and be able to have data injected within the template respectively.

One feature that seems to be most absent, or lacking in strong support in most the languages is I18n(*Internationalization and localization*) support. In the modern Web it seems that having poor I18n support is a huge weakness in a templating language for the Web. According to Alexa estimates Google.com's top five countries are as follows United States, India, Japan, Brazil, and Russia. Out of 5 of the top countries only one of those is an English speaking country, and only two of which are Latin based languages. It is important now more than ever to have strong support for other speaking languages.

1.4 Building a transcompiler.

Building transcompiler is a difficult task. The two primary sources for building a transcompiler will be Compilers Principles Techniques and Tools (2nd Edition) by (Alfred et al. 2007) also known as "*The Dragon Book*", and Parsing Techniques - A Practical Guide (2nd Edition) by (Dick and Ceriel 2008). Since the project is for a transcompiler, and not a traditional compiler, there is a lot within the sources that won't be used. For example The Dragon Book has chapter's such as "Run-Time Environments", and "Machine-Independent Optimizations" which don't relate as are output is HTML, and not Machine Code. The syntax, and the features of the language will need to be mostly finalised early within the project to prevent any sort of feature creep, or spending time, and resources on changing the style of the language. Especially aspects like whether to have natural templates, which are templates which are independent of the language, and have a much more verbose language similar to (*Mustache*), or to have a much more terse language, but have it only work for outputting HTML files, like (*Jade - Template Engine*).

1.5 Why Rust?

There are many reasons that make Rust an great candidate for building a transcompiler. Rust doesn't have a garbage collector that other languages like Java, or JavaScript. This reduces the runtime overhead of the program giving a significant performance boost, which is important for doing on request compilation of templates on a server. Rust's char type is a Unicode scalar value. Meaning Rust can handle non-Latin characters much more easily than languages without IE: JavaScript. Which would be an important feature since the {NAME TBD} will have strong I18n support. Since rust has such a minimal runtime, it has strong FFI(Foreign Function Interface) support, meaning the transcompiler can be used in servers written in different languages, without requiring the compiler to be rewritten in that language, and keeping the performance gains from having it built in Rust.

Chapter 2

Requirements

2.1 Requirements Analysis

The user(developer) should be able to pass in a file written in TBD, and optionally data stored in a key-value pairing. This data will then be passed to a transcompiler, which will return a html file based on the file, and data to be passed back as the HTTP response.

```
doctype html
html
    head
        title= title
        meta( charset="utf-8" )
        meta( name="viewport"
              content="width=device-width, initial-scale=1.0" )
        meta( http-equiv="X-UA-Compatible"
              content="IE=edge,chrome=1" )
        link( rel='stylesheet'
              href='/stylesheets/normalize.css' )
        link( rel='stylesheet'
              href='/stylesheets/skeleton.css' )
        link( rel='stylesheet'
              href='/stylesheets/style.css' )
        script( type='text/javascript'
                src='/javascripts/jquery.min.js'
                defer )
        script( type='text/javascript'
                src='/javascripts/script.js'
                defer )
    body
        block content
```

Figure 2.1: Layout file

```
extends layout

block content
    h1 Hello World

    p Lorem Ipsum...
```

Figure 2.2: Index file

```
<!DOCTYPE html>
<html>
    <head>
        <title>Express</title>
        <meta charset = "utf-8">
        <meta name="viewport"
              content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible"
              content="IE=edge,chrome=1">
        <link rel="stylesheet"
              href="/stylesheets/normalize.css">
        <link rel="stylesheet"
              href="/stylesheets/skeleton.css">
        <link rel="stylesheet"
              href="/stylesheets/style.css">
        <script type="text/javascript"
                src="/javascripts/jquery.min.js"
                defer></script>
        <script type="text/javascript"
                src="/javascripts/script.js"
                defer></script>
    </head>
    <body>
        <h1>Hello World</h1>
        <p>Lorem ipsum...</p>
    </body>
</html>
```

Figure 2.3: Rendered file

2.1.1 Technical Requirements

All the other technologies would be programmed in Rust, and directly interfaced within a Rust program.

Rust

A new programming language from Mozilla, released in May 15th, 2015.The Rust Core Team 2015 Rust is a systems level programming language. Meaning that Rust is run directly on top of the existing OS(Operating System), as opposed to languages like Java, or Ruby which is run on top of a VM(Virtual Machine). Rust is designed to "*combines low-level control over performance with high-level convenience and safety guarantees*" The Rust Core Team 2015.

```
fn main() {
    for i in 1..101 {
        match (i%3, i%5) {
            (0, 0) => println!("FizzBuzz"),
            (0, _) => println!("Fizz"),
            (_, 0) => println!("Buzz"),
            (_, _) => println!("{}", i),
        }
    }
}
```

Figure 2.4: Fizz buzz in rust.

Iron

A high level Web framework designed for making Web servers in Rust.*Iron* Iron is designed to enable other users to create plugins for it. The end-goal would be to integrate the transcompiler with Iron, so a user could easily integrate within the system, and start using the templating language with ease.

```
extern crate iron;

use iron::prelude::*;
use iron::status;

fn main() {
  fn hello_world(_: &mut Request) -> IronResult<Response> {
      Ok(Response::with((status::Ok, "Hello World!")))
  }

  Iron::new(hello_world).http("localhost:3000").unwrap();
  println!("On 3000");
}
```

Figure 2.5: Hello world server using Iron.

2.2 System Model

The system is mainly two parts, the actual Web server written in rust with Iron, and the transcompiler, which will be used as middleware with Iron. Middleware being defined as software that runs between handling the requests, making it easier for the user, or providing new functionality, like adding a templating language.

As shown in TBD. After a request has been handled by the user, but before the response has been sent, the transcompiler will parse the template file, and any file the template requires. Building an AST(Abstract Syntax Tree), and creating a HTML file from the AST. As this is a transcompiler, the program doesn't need to worry about code optimisation, or typical code generation that a typical compiler would. The main work of the program,would be to do lexical analysis, Syntax analysis, and Semantic analysis, and generate the HTML source code if the source is correct, and provide effective, and clear errors if the source code is incorrect.

2.3 Feasibility

There are a lot of risks with making a transcompiler. The most important first step is to have an iron-clad definition of the language it is transcompiling, as changes in the fundamental syntax can cause large sections of code to be rewritten, and could even require in the how the parsing of the syntax is done.

Of course there is always the risk that when the project finishes that it won't have all features specified in the requirements document due to time constraints, but this can be handled with effective time management, and being able to do effective cost analysis such as how viable a feature may be, and how much value does it provide over how much time it will take.

Bibliography

Alex, G. (2014). *Comparing JavaScript Templating Engines: Jade, Mustache, Dust and More*. URL: `https://strongloop.com/strongblog/compare-javascript-templates-jade-mustache-dust/` (visited on 10/26/2015).

Alfred, V. A. et al. (2007). *Compilers Principles Techniques and Tools*. 2nd ed. Addison Wesley.

Chris, W. *Mustache Manual 5*. URL: `https://mustache.github.io/mustache.5.html` (visited on 10/25/2015).

Dick, G. and J. J. Ceriel (2008). *Parsing Techniques - A Practical Guide*. 2nd ed. Springer.

Iron. URL: `https://github.com/iron/iron` (visited on 10/25/2015).

Jade - Template Engine. URL: `http://jade-lang.com/` (visited on 10/26/2015).

Les, O. (2015). *The Verge's web sucks*. URL: `http://blog.lmorchard.com/2015/07/22/the-verge-web-sucks/` (visited on 10/25/2015).

Liam, R. *Transformation*. URL: `http://www.w3.org/standards/xml/transformation` (visited on 10/25/2015).

Mustache. URL: `https://mustache.github.io/` (visited on 10/25/2015).

Sean, W. *How Loading Time Affects Your Bottom Line*. URL: `https://blog.kissmetrics.com/loading-time/` (visited on 10/25/2015).

The Rust Core Team (2015). *Announcing Rust 1.0*. URL: `http://blog.rust-lang.org/2015/05/15/Rust-1.0.html` (visited on 10/25/2015).

W3C and C. James (1999). *XSL Transformations (XSLT) Version 1.0*. URL: `http://www.w3.org/TR/xslt` (visited on 10/25/2015).

Wikipedia, the free encyclopedia. *Comparison of web template engines*. URL: `https://en.wikipedia.org/wiki/Comparison_of_web_template_engines` (visited on 10/26/2015).