Open-Source Technology Use Report

Proof of knowing your stuff in CSE312

Handlebars

General Information & Licensing

Code Repository	github.com/wycats/handlebars.js
License Type	MIT
License Description	• Permission to reuse modify code to suit own needs even if code is proprietary software • License originated at MIT
License Restrictions	• Virtually no restrictions, may reuse and profit off of it. • The only requirement is to include the license and copyright notices in parts of the project where applicable.
Who worked with this?	Patryk Kasza

Handlebars.compile

Purpose

- This method will create a template that we can use to input our own data. It takes a string of html with unfilled data like this {{data}}. It outputs a function that can take a javascript object with data needed to fill the template.
- const templateFun1 = HandleBars.compile(templateHTML1);
    - Line 235 and 263
    - We need to create a template to render our data on the login page and home page.
- var data = templateFun1(templateDict1);
    - Line 236 and 264
    - This is the output function taking in javascript object templateDict1 which will fill the template with the data in the javascript object.

Magic ★★ ° ˙ ° ☽ ° ⁀ ° ★ ≡ ★ ≈

https://github.com/handlebars-lang/handlebars.js/blob/b0f1a62ddb7516694892485c365d97f090b58230/lib/index.js#L7

On the line above we require a folder that doesn't exist in the github repo, or at least I cannot find it. Then after on line 13 of the same file it exports it for us to use. The rest of the lines of code we don't use, as it provides another way of using the handlebars. We can import handlebars to instead export the compile function built within the extension function defined on line 16.

Because of the nature of the official handlebars github repo(not having the files that we actually run, dist/cjs and its contents) I will link the node module from our project repo lines to explain what is going on.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars.js#L39

Here we call:

```
_handlebarsCompilerCompiler.compile(input, options, hb);
```

And we pass in hb, which goes to node_modules/handlebars/dist/cjs/handlebars.runtime.js to create a new runtime object, which calls the function

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars.runtime.js#L40

create() on line 40(called on line 57). Which creates and configures an object out of ->

https://github.com/JunWeiCJW/TeamJJDDP/blob/main/node_modules/handlebars/dist/cjs/handlebars/base.js

Which basically sets up a bunch of helper functions for logging and such.

Returning a HandlebarsEnvironment(from now on as env)

Moving on to executing this function.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L490

Line 491, parses any options we passed through, we don't so we have {}.

Line 493, checks for invalid input. We pass valid so we move on

Line 497, we call _utils.extend({}, options);

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/utils.js#L28

Our code doesn't go past Line 30, as both of our parameters are empty dictionaries, so return an empty dictionary.
Based on line 31, if we did, we could call functions to load values and conditionals to parse for later.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L497

Line 498, empty dictionary so we end up setting the key ‘data’ to true in options

Line 505-533 we set up the function that will be called when we actually parse our template with the values we pass in. Which we will get to later.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/handlerGet.js#L236

We now return the ret function, with sub functions back to our main program in handlerGet.js line 235

Line 236 we call our returned ret function with a dictionary of values that we want to be displayed in our html

Back in compiler.js

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L517

Now since compiled is still undefined(line 505) in this function scope, we call compileInput() on line 523 to assign it to compile.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L485

Now we call env(our HandlebarsEnvironment from before).parse with our input and options(which is just ‘data’:true)

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/base.js#L51

Line 52 takes is to parseWithoutProcessing(input, options);

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/base.js#L33

Line 35 is false as our input is text.

Line 39 to 45 we just assign properties for internal processing of the module. Alongside loading bunch of helper functions to help process on line 30(from yy)

Line 46 is when we call parse from the parser object(line 7-388)

Here we define a bunch of properties to for our parser but most importantly is the fact that is all part of the parser variable which we see does a complex enum matching to switch statements to determine the action of the parser.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L314

This is where the majority of the ‘Parsing is happening’ inside this while loop

The stack holds states which determine the next action.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L342

We see that there are 2 cases being used. The 3rd just returning true.
1 case is indicating that there is a special case found. Regex has been matched with handlebars syntax. And should be added to ‘BlockStatements’

Case 2, handles preparing expression objects and whether or not to add text into the html. ContentStatement object.

On line 320 we our lex function.
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L295

Which then calls the lexer assigned lex function.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L526

Which calls .next()

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L489

Lines 489 to lines 497, load and attempt to match our input(the html file) with rules. Lines 489, mapping to regex expressions.

Once we have a match from the regex, and the index loaded(lines 493-494) we start doing more logic on line 498.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L505

Down to 515 we load and save our findings.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js#L513

We see our input being modified so we don’t match the same thing.

At the end of this we have an array of parts of our original html. Split up into ‘ContentStatements’ or ‘BlockStatements’ to determine the action of the parser when it is applying our data. BlockStatements have a reference to the variable name(under param) that you put in between curly braces to then match with the dictionary that you pass in later, alongside other variables to determine functionality.

So parseWithoutProcessing, does just that. Parses the html we pass in a splits into a format that the parser can then just follow with the values from the defined enum determining its behaviour.

Line 55 in base.js essentially strips the whitespace from our result(ast) variable.

Now we go:
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L509

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L25

Create instance of compiler()

Actually call it here:

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L63

Lines 130-461. These determine ‘actions’ or ‘commands’ to run based on our

Line 64-84 we add functionality through _utils.extend with properties in line 74-82 and then return the new Environment object with new properties based on the ast variable and options.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L84

We call the accept function for the compiler

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L108

We start mapping to types in our program or parsing it itself. Mapping types: 130-461.

We start building the ‘Program’ we will return. Basically ‘mapping’ the program type.

Lines 114 to 127 return a list of commands.
The loop at 118. Accepts each of the items in the program list which maps to different types determining actions for the parser through 130-461.

They are labeled as opcodes, and each of the BlockStatement or ContentStatement returned into our ‘ast’ variable are now being mapped with opcodes.(Based off of lines in 130-461)

We keep going(basically from mapping the array inside the ‘Program’ type) until all is mapped and we get a environment variable with compiler objects(with op codes) to determine how to generate parts of the html.

Those will be called from the main opcode list under the ‘pushProgram’ opcode.

Now going back to:
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L510

Then we create a JavaScript compiler.
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L24

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L73

And then call compile on line 73. Lines 74-88 we load a bunch of values.

Line 88 doesn't do anything since we don't have those options
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L212
this.options.srcName is undefined so it does nothing.

Lines 90 to 97 we define some variables within our compiler object.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L99

Here we call compileChildren.

Now for each child compiler in our environment, we 'compile' it

Line 812 we define a new compiler for the child.

Line 814 is a custom comparator to check if 2 childs are the same name.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L821

This evaluates the child. Following the same steps as we have in the javascript compiler until now. Calling the compile()

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L73

After loading values. Line 88 still does nothing we get to where our environment 'is'. Line 99.

Jumping in
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L99

We have no children, as we are a child, so this does nothing.

Then we jump to lines 110 to 115. Where we evaluate each opcode of our child with its arguments. On line 115.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L115

We end up calling(by mapping) functions defined between 216-799 based on our opcode.

Then we create a function out of our child to return.
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L144

We call it
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L216

Lines 235-239 dont do anything for children.

Lines 239 to 257 is adding/formatting properties to be used when parsing

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L262

This returns a function to run all the opcodes for each child

For the ‘Program’ object containing all the children

We return the ret object/dictionary. That contains our fn variable function.

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-compiler.js#L146

Lines 151 to 182, we load more properties.
Line 186 is false so we go to set some more options defined way before on line 199. And then return on Line 202.

Now we are on
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L511

This jumps us to
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars.runtime.js#L51

Then to
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/runtime.js#L52
We check for valid input and then on 68 we check for pre-compilation. Its false so we go to Line 109 to setup a container with helper functions. Lines 109 to 186

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/runtime.js#L109

We then define a ret function and add ._setup function and ._child function while declaring

the '.isTop' variable to be true. Lines 186 to 256. Then we return the function.

Now we return from compileInput()

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L511

And go back to
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js#L519

We call the ret function then

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/runtime.js#L193

This sets up a bunch of values in our container which we will run

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/runtime.js#L195

Here we initialize our data with the context we passed through(Aka variables to replace on frontend)

https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/runtime.js#L343

We map our content to the root and map any data that we need to by creating frames
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/utils.js#L110

Then we go to line 211.
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars/runtime.js#L211

We don't have an decorators so we go 212.

Here we jump to 208 where we go to the custom VM file that we created:
https://github.com/JunWeiCJW/TeamJJDDP/blob/a8f5590c7fa0647045b553a08b809ff48c2dfe1e/node_modules/handlebars/dist/cjs/handlebars.runtime.js#L49

where we see all our previous contentStatements and BlockStatements now visualized into strings a conditionals/functions respectively.

Whats left is executed the final return value in our VM, appending the result of our "BlockStatement" objects where we have special functionality, we use our opcode mapped variable function and logic derived helperfunctions to handle cases with no data or other edge cases.

And then we get our final, compiled html string.