International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1524-1528 www.ijaem.net ISSN: 2395-5252

A One-Pass JSON compiling technique

Nikhil Kumar Y.
Student,Dronacharya College of Engineering,Gurgaon, Haryana

Submitted: 25-06-2021 Revised: 06-07-2021 Accepted: 09-07-2021

ABSTRACT: JavaScript Object Notation(JSON) open standard file format and data interchange format. It uses human readable text to store data. This paper introduces a compiling technique that is lightweight to implement and compiles it into a C – family data structure using code reflection. Such a software is called JSON parser[1].The implantation of JSON parser described will help in parsing JSON files whose structure is previously known and those files whose structures is unknown.
KEYWORDS: JSON, JSON Parser, C – family languages (C++, Java, etc), Compiling, Parsing, Compiler implementation, Code Reflection.

I.INRODUCTION

JavaScript Object Notation(JSON) open standard file format and data interchange format. It uses human readable text to store data. JSON was based on JavaScript but it is language independent data format. JSON text is serialized[1] meaning order of value matters. This compiler will compile the data while on the fly i.e. while the program using the data is being executed and it will also multiple output formats in language being compiled to depending upon weather the structure of the data is known or unknown. Just like a standard compiler the JSON file will be first processed by the front end of the compiler i.e. The JSON data will be stored in a symbol tree and the back end of the compiler will be described for two conditions. One where the structure of the data will be known and other where structure will not be known. One part of the compiler will be using Code Reflection features of the language. While other will present the data in a list.

JSON represents data as a sequence of tokens. There are six such tokens:

I. Structural Characters
II. Strings
III. Numbers
IV. Literals

The Structural Characters include six characters:
Left curly bracket: - '{'
Right curly bracket: - '}'
Left square bracket: - '['
Right square bracket: - ']'
Colon: - ':'
Coma: - ','
They are used to represent objects or array
Object are data stored as key/value pairs
Their regular expression is:
{ (member,)*+member}
Where,
member = key : value
Key = String
Value = Object|Array|String|Literal|Numbers

Array is used to represent a sequence of values
With regular expression:
[(value,)*+value]
Strings are sequences of characters beginning with quotes their regular expression being
Numbers are sequences of digits they may have fractions or exponents and represent decimal numbers with base 10.
Literals have three values: true, false, null.

A Json file may contain any of these data structures comprised of the 4 tokens.

Since Json file stores the data the front end of the compiler will only analyse the tokens and convert them into values it will require lexical analysis parsing and sematic analysis. The JSON file can contain any of the four types of data object, array, string. An internal data structure should be used to store the values of the JSON file to help with the locality of reference during program execution. This will improve cache performance. The internal data structure used may depend of weather we know the structure of the JSON file or not. In-case the structure of the JSON file is not known Lists can be used to represent the file if it is known the known data structure can be used to produce the output.

II. FRONT-END OF THE COMPILER

In the front end of the compiler lexical analysis , parsing and semantic analysis will be done. The characters in the program will be read by the lexical analyser and will be tokenised i.e. converted into tokens these tokens will be passed

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1524-1528 www.ijaem.net ISSN: 2395-5252

on to the parser to validate if they are following correct syntax. and will be further passed on to semantic analyser which will have a table associating elements of code to particular tokens. The code will differ according to weather the structure is known or unknown.

Implementation of the Lexical analyser

The lexical analyser will work based on a table of regular expression based on which tokens will be generated. Some characters will have associated piece of code by the table in order to help in generating token. These code will have to be associated with generating tokens for Strings, Numbers, and Literals.

The table will be

Character	Associated code function	Token type
{|}|[|]|:|,	Move to next phase	{|}|[|]|:|, Respectively (Token type will be the character itself)
“	Generate string until “ is encountered	String
1-9	Generate digit until ’,’or’]’or’}’ is encountered	Number
t	Check if true and generate token true	Literal
f	Check if false and generate token false	Literal
n	Check if null and generate token null	Literal

Further more a enumerable variable storing class of will be used to simplify parsing. This variable will store Token type. Particular token types are shown in above table. From here the token will be passed to the parser.

Implementation of the parser

The parser will work based on the parsing table(Table 1)a stack and a variable current_state. Current state is initialised to 0.The parsing table is generated usingstate diagram[2].The column of the parse table represents a state and each row represents a token. Each token have a command associated to it for every state. The commands being:

Goto(state):setcurrent_state to state
Push(state):push the state into the stack
X:error return error
return: return output
Pop(): pop the stack and use the element returned as current_state

In case of return error there is an error in the file.After every set of commands are executed and if the commands are not “return output” or “error” the state value ,token, and it’s value is passed to the semantic analyser.

Implementation of semantic analyser

The semantic analysis have a table similar to parsing table stored in it this table have commands to perform during state transition and each command is stored in list that have both a stack and list like interface. This list take a pairas it’svalue and stores it. These include name of current key and binary value indicating weather the value is being stored in a object or an array. This will help in reconstructing the name of the array or object. Beside the parsing table semantic analyser also uses a Key field and current state denoting whether the current item is a object or an array.

Along these variables a few functions are used to achieve the last part the definition of these functions vary depending upon whether we know the structurer of the JSON or not. When the values are known these functions are:

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1524-1528 www.ijaem.net ISSN: 2395-5252

GetName():This function traverses the stack and constructs the name using the following method. For i^{th} element in the stack add name of the element to the name being computed
if second variable of the stack is true and append it with a '.'.ie if the element in the stack is
[abz , true] append current name with "abz."
If the second variable is false add the prefix '[' before adding the name and follow it with the suffix ']' I.e. if the element is
[3,false] append current name with "[3]."

This method will generate the name coupled with code reflection this method will point to wards the memory location of the name that is generated. Replace the value in the memory location of the name with value in token_value
i.e. value of the token
Push(Key, true/false) will push values of key and true or false representing object or array.

	0	1	2	3	4	5	9	10
String	Goto(1)	X	Goto(3)	X	5	X	Goto(10)	X
Number	Goto(1)	X	X	X	5	X	Goto(10)	X
Literal	Goto(1)	X	X	X	5	X	Goto(10)	X
{	Push(1) Goto(2)	X	X	X	Push(5) Goto(2)	X	Push(10) Goto(2)	X
[	Push(1) Goto(9)	X	X	X	Push(5) Goto(9)	X	Push(10) Goto(9)	X
}	X	X	X	X	X	Pop()	X	X
]	X	X	X	X	X	X	X	Pop()
:	X	X	X	Goto(4)	X	X	X	X
,	X	X	X	X	X	Goto(2)	X	Goto(9)
EoF	X	return	X	X	X	X	X	X

(Table 1)

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1524-1528 www.ijaem.net ISSN: 2395-5252

	0	1	2	3	4	5	9	10
String	Value= token..value		Key= token.value		Value(GetName()) = token.value.		Value(GetName()) = token.value.	
Number	Value= token.value				Value(GetName()) = token.value.		Value(GetName()) = token.value.	
Literal	Value= token.value				Value(GetName()) = token.value.		Value(GetName()) = token.value.	
{	Push(Key, true)				Push(name, true)		Push(name, true)	
[	Push(Key, false) Key =0				Push(Key, false) Key =0		Push(Key, false) Key=0	
}						Currentname =Pop( ).first		
]								Currentname =Pop( ).first
:								
,								Key ++
EoF		Return Value.			\			

(Table 2)

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1524-1528 www.ijaem.net ISSN: 2395-5252

(Continuation from page2) Pop() will recover the value of key and binary value indicating array or object was being stored.
The above method when used to parse JSON parses the data in the JSON file into a data format present in the language

III.CONCLUSION

The outcome of the theorised parser is that a lightweight parser can be implemented in any of the c – family languages. Such an implementation is important for may web based application written in popular languages such as java ,c# and c++.These implementation will provide the said languages an easy method to access data present in the web without much effort.

REFFERENCES

[1]. Internet Engineering Task Force (IETF), Request for Comments: 8259 https://datatracker.ietf.org/doc/html/rfc8259
[2]. Compilers - Principles, Techniques, and Tools-Pearson_Addison Wesley (2006), Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman,ISBN 0-321-48681-1